

Applicability of Transfer Learning on Sinhala Named-Entity Recognition

Akuratiya Gamage Kavisha Chiranjaya
Abeynayaka



Applicability of Transfer Learning on Sinhala Named-Entity Recognition

Author

A.G.K.C Abeynayaka

Index Number: 20000049

Supervisor: Mr. Viraj Welgama

Co-Supervisor: Mrs. A.L. Nanayakkara

April 2025

Submitted in partial fulfillment of the requirements of the
B.Sc in Computer Science Final Year Project (SCS4224)



Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: A.G.K.C Abeynayaka



Signature of the Candidate

Date: 28-04-2025

This is to certify that this dissertation is based on the work of Mr. A.G.K.C Abeynayaka under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Principle Supervisor's Name: Mr. Viraj Welgama



Signature of the Supervisor

Date: 28-04-2025

ABSTRACT

Named Entity Recognition (NER) is a preliminary task in Natural Language Processing. NER has evolved from relying on rule-based mechanisms to utilizing neural networks. NER is a pretty much resolved matter in the English language. The Sinhala language faces the issue of data scarcity due to its complexities with dataset extraction. Manual annotation of a Sinhala-labeled dataset is a laborious task.

Entity recognition solely depends on a tagged dataset in a specific language, but due to data limitations, it's hard to do experiments on NER models in Sinhala. However, most of low-resource NLP researches shows remarkable improvement with the knowledge transferring mechanism, which is known as transfer learning. This research suggests a Sinhala NER model based on transfer learning, considering monolingual and multilingual approaches. An Indic language model is fine-tuned for the target Sinhala NER model during both approaches. The IndicBERT(Kakwani et al. 2020) model is chosen as the source model due to its similarity with Sinhala.

The evaluations were done on monolingual and multilingual datasets. For the monolingual dataset, a separate dataset was created using a weakly supervised automatic method that contains six different categories. The multilingual dataset was created with a Bengali dataset. The final transfer learning model was trained on hyperparameter tuning followed by an augmented dataset from monolingual data. It showed a moderate precision of 48.21%. The baseline CRF model showed a macro precision of 90% and a macro F1-score of 61% showing that CRF is applicable in normal contexts.

Keywords: Sinhala, Named Entity Recognition, Transfer Learning, IndicBERT, CRF

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Mr. Viraj Welgama, for his invaluable guidance, knowledge, and encouragement during the whole length of this project. I am deeply appreciative of Ms. Lakshika Nanayakkara for her invaluable perspectives and unwavering assistance.

I express my heartfelt thanks to Mrs. Alok Fernando and Dr. Surangika Ranathunga of the University of Moratuwa for their guidance and support, and especially for providing a part-of-speech tagger for my research. I am also grateful to Aravind Krishnan of Saarland University of Germany for providing insights into implementation.

Finally, I would like to express my heartfelt gratitude to my parents, who helped me through tough moments in my life.

Contents

Declaration	1
Abstract	2
Acknowledgements	3
Acronyms	6
1 Introduction	9
1.1 Background	9
1.2 Problem Statement	11
1.3 Research Aim, Questions and Objectives	12
1.3.1 Research Aim	12
1.3.2 Research Questions	13
1.3.3 Research Objectives	15
1.4 Significance of the Project	16
1.5 Research Approach and Methodology	16
1.6 Scope and Delimitations	18
1.6.1 In scope	18
1.6.2 Out scope	19
1.7 Outline of the Dissertation	20
2 Literature Review	21
2.1 NER for Low resource languages	21
2.1.1 NER for Sinhala Language	25
2.2 Transfer Learning for low resource languages	28
3 Design	35
3.1 IndicBERT	35
3.2 CRF	36
4 Implementation	40
4.1 Dataset Preparation	40
4.2 Implementation of the Baseline NER model	49
4.2.1 Splitting the NER dataset	49

4.2.2	Implementation of the monolingual CRF model	50
4.3	Implementation of the Monolingual Transfer Learning Model . . .	54
4.3.1	Data Augmentation	59
4.3.2	Hyperparameter Tuning	61
4.4	Implementation of the Multilingual Transfer Learning Model . . .	62
5	Results and Analysis	65
5.1	Evaluation Metrics	65
5.1.1	Precision	66
5.1.2	Recall	66
5.1.3	F1- Score	67
5.2	Experiments and Results on the Monolingual Model	67
5.2.1	Results from the baseline model	67
5.2.2	Results of the monolingual transfer learning model	69
5.3	Experiments and Results on the Multilingual Model	71
6	Conclusion	73
6.1	Summary	73
6.2	Limitations	75
6.3	Future directions	75
	References	77
	Appendices	84

Acronyms

CNN - Convolution Neural Networks

CoNLL - Conference on Computational Natural Language Learning

CRF - Conditional Random Fields

F1 - F1 Score (a measure of a test's accuracy)

GloVe - Global Vectors for Word Representation

HMM - Hidden Markov Models

IOBES - Inside Outside Beginning End Single

LSTM - Long Short-Term Memory

mBERT - multilingual BERT (Bidirectional Encoder Representations from Transformers)

ME - Maximum Entropy

MUC-6 - Sixth Message Understanding Conference

MuRIL - Multilingual Representations for Indian Languages

NER - Named Entity Recognition

NLP - Natural Language Processing

POS - Part of Speech Tagger

RNN - Recurrent neural networks

RQ - Research Questions

Word2Vec - Word to Vector

XLM-R - Cross-lingual Language Model - RoBERTa

List of Figures

2.1	Traditional Machine Learning vs Transfer Learning (Pan and Q. Yang 2010)	28
2.2	Study of graded addition of languages for NER task on low resource languages of(left) Oriya and(right) Punjabi using MuRIL(Dhamecha et al. 2021)	30
2.3	state of the art results for Spanish and Dutch NER (Z. Yang, Salakhutdinov, and W. W. Cohen 2017)	32
2.4	CNN Bi-LSTM model for NER used by (Murthy and Bhattacharyya 2018).	33
2.5	Comparison of Various Multilingual Learning Strategies and Monolingual Deep Learning Systems on Tamil, Malayalam, Bengali, and Marathi NER Using Hindi as Assisting Language	34
3.1	A layout of the BERT-CRF model.	37
4.1	Data flow diagram of the data set generation process(Krishnan et al. 2021)	42
4.2	Input file format for the POS tagger.	47
4.3	Model files for the Sinhala POS tagging	48
4.4	POS tagged tokens in .tts format	48
4.5	Gazetteer list used in the Research	51
5.1	An example of token-level tagging scheme	66
5.2	An example of entity-level tagging schemes	66
5.3	Baseline CRF model performance	69

List of Tables

4.1	Distribution of train, dev and test dataset	50
5.1	Comparison of Macro-Averaged Precision, Recall, and F1-Score across three evaluation instances	68
5.2	Best test set performance based on precision in a monolingual transfer learning	70
5.3	Best test set performance based on precision for transfer learning in a monolingual model with augmentation.	71
5.4	Best test set performance based on precision for transfer learning in a multilingual model.	72

Chapter 1

Introduction

1.1 Background

The natural language processing helps computers communicate with humans in their native language. For example, NLP makes it possible for computers to read text written in their own language, recognize speech, interpret it, measure sentiment, and determine which parts are more important for analysis. The Name Entity Recognition (NER) is a form of NLP data processing task, and it involves the identification of key information or entities in the given text and classify them into predefined categories like person (PER), organization (ORG), place/location (LOC) etc.

Named Entity Recognition has different applications in various fields, such as health care, social media, and entertainment. The most common applications of NER are information extraction, where we mainly focus on identifying the most important entities from the given unstructured text and labeling them as person, organization, and location. Through NER, search engines can boost their searching abilities with a better understanding of the context of the query and lead to more accurate and relevant results. In addition to that, social networks can use NER to analyze content or posts and recommend similar articles, brands, or products.

The term **NER** was first coined in the Sixth series of Message Understanding Conference (**MUC-6**) in 1996. After that research work began around NER in the English language. At the beginning, most of the research work revolved around using rule based methods such as regular expression matching, heuristic

rules, and String matching. But rule based methods were very time-consuming, and they required linguistic expertise as well. Not only that, but rule-based NER is more suitable for controlled environments with well-defined entities. However, most NLP applications operate in adaptive environments that learn by analyzing information from the given inputs. Therefore, machine learning-based approaches have been introduced to cater to a wider range of inputs. This is because the accuracy of NLP applications relies heavily on the richness of the desired dataset.

After like 2000 machine learning algorithms like Hidden Markov Models (HMM) (Zhou and Su 2002), Conditional Random Fields (CRF) (N. Patil, A. Patil, and Pawar 2020), and enhanced SVM approaches (T. D. Singh, Basanta, and P. K. Singh 2020) revolutionized Named Entity Recognition through supervised learning. The arrival of deep neural networks made an impossible breakthrough in English NER, which works perfectly on large datasets in a more accurate way. Deep neural network (DNN) models use a method called representation learning to automatically learn patterns and features from input data. This approach differs from traditional handcrafted feature engineering(**HMM,CRF**), where humans manually design and extract important features from the data before feeding them into a model.

There are different deep learning architectures for NER, such as transformers, BiLSTM-CRF, end-to-end neural NER, and CNNs (Lample et al. 2016). However, in Deep Learning architectures like BERT, it can be adapted for Named Entity Recognition (NER) tasks because of its features. BERT models are normally pre-trained by using large corpora, which allows them to identify language-dependent patterns easily. These types of pre-existing models are widely useful with low-resource NER model training, especially when there is limited labeled data due to language complexity. Most low-resource languages face challenges in collecting datasets because of their complexities, which can be effectively addressed using a transfer learning approach. In addition to that, BERT models can understand the context of a word from both directions (left and right) with

the help of adding some **Bi-LSTM** layers, which ultimately helps to accurately identify the entities based on the meaning of words in their context. As a result, BERT models are the most popular approach for NER model development. In Deep Learning architectures like **BERT** can be adapted for Named Entity Recognition (NER) tasks. Here usually in practice BERT model is combined with a decoder layer for better accuracy. While models such as **RoBERTa** are primarily trained on English data, **XLNet-RoBERTa** is pre-trained on multiple languages, making it suitable for Fine-tuning with low-resource languages.

Low level languages face the inherent challenge of data scarcity. But we can use multilingual models like **XLNet-RoBERTa** to tackle this issue by using transfer learning. Transfer learning is a machine learning method where the knowledge learned from one task is applied to another separate or the same but related task (Tan et al. 2018). Since transfer learning allows us to use a Pre-trained model, we can reuse the model with a low amount of labeled data .

Anyway, NER gives much more accurate results for the most widely used high-resource languages, likes English, Spanish, French, and German. Low level languages face the inherent challenge of data scarcity. But we can use multilingual models like XLNet-R to tackle this issue by using transfer learning. Transfer learning is a machine learning method where the knowledge learned from one task (source model) is applied to another separate or same but related task (target model) (ibid.). Since transfer learning allows us to use a Pre-trained model we can reuse the model with a low amount of labeled data. It showed performance improvement on low-resource NLP applications such as ASR, POS taggers, and NER.

1.2 Problem Statement

The problem statement is focused on finding out the applicability of the transfer learning approach on monolingual and multilingual NER for the Sinhala language. Sinhala language is the native language of Sri Lanka and it belongs to

Indo-European language family. The Current state of the art work in Sinhala NER involves using machine learning algorithms like HMM and CRF inspired from other Indic language resources (J. Dahanayaka and R. Weerasinghe 2014). However machine learning algorithms face the inherent data scarcity issue, which is more common in low-resource languages like Sinhala, that hinders the pattern recognition from data.

But there are very few attempts to explore the effectiveness of transfer learning in the Named Entity Recognition on the Sinhala Language. However, they failed to perform effectively on entity recognition; due to that fact, its applicability to Sinhala NER remains unexplored. That creates a significant gap in Sinhala NER research, creating difficulties in entity recognition competent with the accuracy of data-rich languages. Therefore, it is crucial to investigate and mitigate that gap by using the knowledge extracted from a well-resourced language pretrained NER model. Since now there are existing pre-trained multilingual Indic models such as Indic-BERT (Kakwani et al. 2020), through that we can exploit the shared patterns or linguistic features from the language which are similar to Sinhala such as Bengali, which could be helpful in compensating for the data scarcity issue

1.3 Research Aim, Questions and Objectives

1.3.1 Research Aim

The main aim of this research is to identify how transfer learning can impact entity recognition in the Sinhala language. Through this research, we are able to address the challenges of data scarcity for Sinhala by considering both monolingual and multilingual approaches. As a result, this research can enhance the performance of Sinhala NER.

1.3.2 Research Questions

RQ1: What are the most effective parameters for enhancing Sinhala NER using a transfer learning approach?

To enhance the performance of Named Entity Recognition (NER) for Sinhala using a transfer learning approach, it is necessary to focus on hyperparameter tuning. First, we have to select the appropriate pretrained model and then fine-tune the model with the Sinhala dataset, optimizing hyperparameters such as learning rate, batch size, and number of epochs. Since our selected pre-trained model is trained for Indic languages, all the parameters were identified based on the features of that particular language, which may not be compatible with our Sinhala dataset. Therefore, it is essential to determine the optimal combination of hyperparameters for the selected Sinhala dataset.

A pre-trained model is tuned with selected hyperparameters to see the optimal combination of hyperparameters and the weights of the model.

RQ2: How does the application of different data augmentation techniques improve the overall performance of the entity recognition in the Sinhala NER?

The application of different data augmentation techniques has an impact on the performance of named entity recognition in Sinhala by improving the volume and variations of training data. Since Sinhala is a morphologically rich, complex language, detecting large variations in data coverage is a time-consuming task. It is essential to use data augmentation to mitigate that problem. Existing augmentation techniques such as synonym replacement, back-transliteration, and random insertion will enhance the existing data variations, which ultimately leads to better results. By analyzing the wider range of data variations of linguistic patterns, the model can effectively learn entity identification in an accurate manner, which ultimately improves the model’s accuracy. Therefore, data augmentation is necessary for low-resource languages like Sinhala, where annotated datasets are limited. The best model selected after the monolingual and multilingual training

will be trained again with augmented data. Only the native sinhala language will be subjected to augmentation .

RQ3: In what ways do monolingual and multilingual transfer learning techniques impact the performance of NER models specifically for the Sinhala language?

Monolingual and multilingual transfer learning techniques significantly impact the performance of low-resource NER models. Monolingual transfer learning focuses on using data and models that were trained on Sinhala, which can help enhance the model’s accuracy and entity recognition relevant to the Sinhala language and its morphological structure. This approach allows the NER model to predict the named entities based on the specific linguistic features in an accurate manner. On the contrary, multilingual transfer learning helps to mitigate the data scarcity issue in low-resource languages. Here, it extracts language features from multiple languages that are similar to our target language, Sinhala. By training on language data related to Sinhala, which belongs to the Indo-Aryan family of languages, such as Bengali, the model can gain insights and linguistic patterns that are hard to obtain from Sinhala alone, thereby enhancing its ability to recognize entities from the given text. However, the accuracy of NER can be different depending on the selection of linguistic similarities and the languages chosen. Therefore, it is necessary to find out the impact of both monolingual and multilingual transfer learning models, as they can shape the NER model’s performance in entity recognition for Sinhala.

The idea is to experiment with the model training with multilingual data. Bengali is another Indo-Aryan language, just like Sinhala. Multilingual data and Sinhala label data is used together to train the model. The data scarcity issue is addressed from a different dimension here.

1.3.3 Research Objectives

RO 1: Investigate the effectiveness of transfer learning on Sinhala NER by comparing the performance against a baseline model.

RO 2: Experimenting on Data Augmentation to see whether there is an improvement in Pre-trained multilingual model performance.

RO 3: Experimenting with monolingual and multilingual fine-tuning to see whether there is an improvement in the performance of the Pre-trained multilingual model.

RO 4: Creating a dataset with more fine-grained categories than the traditional PER, LOC, and ORG classes, using a suitable automatic method.

RO 5: Experimenting with different selected parameter combinations to identify the most effective set for transfer learning in Sinhala NER.

1.4 Significance of the Project

NER is a crucial component of NLP systems that can greatly benefit Sinhala language processing when it comes to information retrieval, searching and content classification. Sinhala is a language with its own unique characteristics and challenges, such as extensive vocabulary, complex grammar rules, phonetic structure, and ambiguity, and NER can help address some of these challenges by accurately identifying and categorizing named entities in Sinhala text.

Virtual assistants and AI chatbots are increasingly used to interact with users in their native language. NER plays a vital role in these systems by enabling them to understand user queries more accurately. For example, by identifying entities like names, locations, and organizations, virtual assistants can provide more context-specific responses, leading to better user experiences. NER can also be valuable in the newspaper industry for organizing articles based on named entities mentioned within them. By automatically identifying and categorizing entities, newspapers can group related articles together more efficiently. This enhances the browsing experience for readers, allowing them to access relevant content more easily.

The significance of NER extends beyond virtual assistants and newspapers. Many other NLP tasks rely on accurate entity recognition, such as information extraction, document summarization, sentiment analysis, and more. By improving NER performance in Sinhala, the research has the potential to enhance various aforementioned NLP applications.

1.5 Research Approach and Methodology

- **Recreation of the Baseline Models and provision of a Dataset**

- Most of the existing datasets are limited to the three main categories: **LOC**, **PER**, and **ORG**. However, we have implemented a novel dataset for the Sinhala language that was extracted from Wikipedia

using a weakly supervised method covering six categories: **Location (LOC)**, **Creative Work (CW)**, **Organization (GRP)**, **Person (PER)**, **Product (PROD)**, and **Medical (MED)**. The detailed explanation is provided in the Implementation chapter.

- For the purpose of multilingual training, we utilize the public dataset (Fetahu, Z. Chen, et al. 2023) used in the work of (Fetahu, Kar, et al. 2023). This dataset is in Bengali and includes seven fine-grained categories that correspond to those in the aforementioned monolingual dataset.
- We used the POS tagger introduced by (Fernando and S. Ranathunga 2018), along with the tag-set developed by (Fernando, S. Ranathunga, et al. 2016), for the POS tagging task. Because it will improve the model’s understanding with respect to the linguistic features and lead to more accurate entity recognition.
- The model introduced by Azzeez and Ranathunga (Azzeez and Surangika Ranathunga 2020a) is re-implemented from scratch, using the same set of features as the original, and it was trained with a previously extracted dataset. It was used as our baseline model for evaluations.

- **Search for appropriate Pre-trained models**

The next step of the research is to search for pre-trained models in order to experiment on transfer learning. Because in transfer learning it will do the prediction on the target Sinhala language using the data-rich source model, and depending on the source model, it will vary the model performance. So it is necessary to choose the most suitable model, which is compatible with different kinds of language combinations.

- Languages that are related to Sinhala but not very rich in resources.
(E.g. - Hindi and other Indo-Aryan languages)

- **Implementation of Multilingual Models**

As the next step, multilingual models will be trained using the chosen pre-trained models. And they will be compared to obtain the best pre-trained model suitable for Sinhala in terms of accuracy and resource availability. Firstly, the models will be trained with random parameters, and then hyperparameter tuning is applied to observe any significant improvement in performance.

- **Experimenting with monolingual, multi-lingual training and employing data augmentation techniques to come up with a optimal sinhala transfer learned NER model**

- Training the model with multiple languages (Bengali + Sinhala) and from Sinhala, respectively, using the transfer learning approach from the IndicBERT model to predict NE tags in multilingual and monolingual approaches.
- Training models initially with the annotated dataset and then again with the augmented datasets. Here the expectation is to augment the Sinhala dataset to cover more depth coverage.
- Finally evaluate the model’s performance using the matrices as described in the 5th chapter.

1.6 Scope and Delimitations

1.6.1 In scope

- Exploring the different approaches of named entity recognition systems for low-resource languages.
- Exploring the various transfer learning methods used in named entity recognition systems for low-resource languages, categorized into three groups.

- Exploring the different approaches of dataset extraction using Wikipedia articles used in named entity recognition tasks.
- Collecting a multilingual dataset.
- Collecting existing named entity datasets for Sinhala.
- Extracting a new dataset for Sinhala using Wikipedia articles, covering six categories.
- Implementing a baseline CRF-based named entity recognition model for the Sinhala language covering six categories.
- Identify the best-suited target model with hyperparameter-tuned values.
- Analyzing and fine-tuning the target Sinhala model using data augmentation techniques.
- Fine tuning the model with a multilingual dataset.
- Comparing the results with existing NER models for Sinhala.
- Fine-tuning pre-trained multilingual models using data augmentation.
- We chose the IOBES tagging format for annotating datasets.

1.6.2 Out scope

- All 23 categories in the reference work won't be considered here.
- Nested entity ambiguity detection is not addressed.
- Data augmentation is only relevant for Sinhala NER.
- Multilingual models are used only for feature extraction purposes .
- For multilingual training, we only use the Bengali language.
- The final goal is to build a Sinhala NER model to detect more categories than the existing 3-category (**PER,LOC,ORG**) model.

1.7 Outline of the Dissertation

The dissertation is organized as follows. In chapter 2, a literature review has been conducted to indicate the gap between the current state of the art Sinhala NER and other NER systems in terms of the technologies used. Chapter 3 contains in-depth information on the technologies and research methodology used, including the corresponding architectures and algorithms used in this research. Chapter 4 outlines the step-by-step approach taken in this research has been outlined. Chapter 5 displays the results from the experimentation conducted throughout the research and an analysis of the results. Finally, Chapter 6 provides the conclusions drawn from the research and discusses future work.

Chapter 2

Literature Review

2.1 NER for Low resource languages

Natural languages can be classified into two broad categories, i.e., low-resource languages (LRLs) and high-resource languages (HRLs). For high-resource languages, many data resources exist that help to enable machines to learn and understand natural languages, e.g., English. English is a well-resourced language as compared to other spoken languages. Many Western European languages are well-resourced languages. Chinese, Japanese, and Russian are also high-resource languages. In contrast, low-resource languages have very few or no resources available. Low-resource languages may be described as less studied, resource-scarce, less computerized, less privileged, less commonly taught, or low-density languages (Cieri et al. 2016). Although data-rich models like English or German can accurately identify the named entities due to their linguistic coverage, it is challenging for us to integrate directly with low-resource languages like Sinhala for NER tasks because of linguistic differences (Manamini et al. 2016). Under this section, some of the existing low-resource NER research was critically analyzed, providing a summary of their works, including results, approaches, datasets, and observations. Different NER approaches involving other low-resource languages are discussed in a detailed manner.

In 2018, researchers delved into some way of improving the performance of the neural network in NER tasks using a soft gazetteer method instead of traditional gazetteer features (Wu, Liu, and Cohn 2018). This study focuses on improving neural network performance on NER tasks using (ibid.) a soft gazetteer method

instead of traditional gazetteer features. They challenge the existing belief that hand-crafted features are unnecessary for deep learning models because they will learn knowledge automatically from their corpora. Their implemented hybrid approach shows that integration of manual features such as part-of-speech tags, word related features and gazetteers can improve the performance of the Neural-CRF model, obtaining a 91.89 F1 score for the CoNLL-2003 English shared task. Through this approach it will not simply enhance the performance of entity recognition but also reduce the training requirements by 60% which is significant compared to other baseline models.

A gazetteer simply refers to a list of categorized entities such as organizations, person, locations, days of the week, etc. It helps to properly identify the entities from the given text. Apart from traditional gazetteers, soft gazetteers are considered a more flexible entity recognition technique. Because it will consider variations of text such as misspellings and synonyms rather than simply checking for existing matches. So, in NER, soft gazetteers can enhance the model's entity recognition accuracy by considering additional information.

Wu, Lin and Cohn have been introduced an approach with hand-crafted features for entity recognition, which impossible with low-resource languages. Therefore, Rijhwani et al. proposes a novel approach to improve the NER model performances in low-resource languages such as Kinyarwanda, Oromo, Sinhala, and Tigrinya, where it's hard to find lists of entities in those languages (Rijhwani et al. 2020). Their approach relies on the concept called "soft gazetteers," which Soft gazetteers incorporate ubiquitously available information from English knowledge bases, such as Wikipedia, into Neural-Named Entity Recognition models through cross-lingual entity linking.

Given an input sentence, soft gazetteer features for each word can be denoted as $s = w_1, \dots, w_n$, and then the features can be applied to each word in the span. It is assumed that there is an entity linking (EL) candidate retrieval method that returns candidate knowledge base (KB) entries considered as $C = (c_1, c_2, \dots)$

for the given input span, where c_1 is considered as the highest-scoring candidate among others. Low-resource languages used in the research are Kinyarwanda, Oromo, Sinhala, and Tigrinya.

The baselines for this study are CNN-LSTM-CRF (NOFEAT: a model without considering any features) and BINARYGAZ, which uses Wikipedia entity lists to create binary gazetteer features. First, comparing BINARYGAZ to NOFEAT shows that traditional gazetteer features help somewhat, but gains are minimal on languages with fewer available resources. Advanced soft gazetteer methods such as PBELSUPER (supervised learning-based method) and PBELZERO (zero-shot transfer method) are used in the research as soft gazetteer methods.

The model’s evaluation is based on 10-fold cross-validation due to its data scarcity, and it considers the F1 score as the primary evaluation metric. PBELZERO method slightly improves the F1 score for Kinyarwanda, Oromo, and Tigrinya from the BINARYGAZ, indicating that these methods can effectively enhance NER by leveraging information from related languages. While PBELSUPER (trained on the small number of bilingual texts) improves the F1 score of Sinhala from 54.08 (BINARYGAZ) to 60.95 (PBELSUPER). Despite all of this, both ORACLEGAZ and ORACLEEL, known as artificially strong systems, improve NER performance of all languages significantly by hitting the NER F1 scores with over 90, exceeding all non-oracle methods, indicating that there is substantial space to improve low-resource NER through either the development of gazetteer resources or the creation of more sophisticated EL methods.

The impact of soft gazetteer features is further emphasized by incorporating these features, which increases recall (the ability to identify true positives correctly) for both non-NIL with linking and the unseen category in the training set mentions and unseen mentions by 5.5 and 7.1, respectively, by several points compared to the baseline. However, these improvements are limited by the number of entities that the KB covers. Consequently, they augment entity mentions in the KB. This "augmentation" of the KB means they include additional NIL

mentions. When they do this, the performance of the soft gazetteer methods improves significantly for all four languages, achieving the highest improvement of 9.97 in the F1 score for Sinhala among other languages(Rijhwani et al. 2020).

Another study focuses on using the novel LLM to solve the NER recognition problem. Here the use case of LLM is vastly different from its original task, which is the generation of text, as clearly stated in the text. Hence the result won't be that promising. But the writers suggest a method to transform the NER task to a text generation task, which more adapted to LLM tasks. For example the input **“Colombo is a city”** will be marked as **“@@Colombo is a city”**. The results certainly abnormal compared to annotated corpus but since this is baseline research work, the format of the output is acceptable(S. Chen et al. 2021) .

Recently, in 2023, in this particular study mainly idea is to leverage high-resource language to tackle low-resource language learning problems . This leveraging process is also known as cross-lingual transfer learning in NLP, which uses knowledge from multiple languages. The authors emphasize the enhancement of transfer learning model training efficiency and its performance on entity recognition while keeping the source model's weights and size unchanged for the target model predictions. This adds a new dimension to transfer learning. Rather than solely cross-lingual-transferring or fine-tuning pretrained models, the authors try to create a significantly similar-sized model from a source to a target language, which reduces resource consumption by 80% while maintaining the highest accuracy on entity recognition(Ostendorff and Rehm 2023).

Another somewhat different study(Krishnan et al. 2021) focuses on employing the public Wikipedia source to provide a scalable and practical way to solve the NER problem. The issue of lack of annotated data is addressed by a weak-supervised method using both Wikipedia and Google Knowledge Graph to bootstrap NER data without any manually annotated data. The authors design an adaptive method which suits morphologically complex languages. This method is tested in 2 different but morphologically complex, agglutinative languages called

Malayalam and isiZulu. Further The system’s performance was evaluated on both in-domain (Wikipedia-derived) and out-of-domain datasets. The XLM-R model showed significant in-domain performance for Malayalam ($F1=0.87$) and isiZulu ($F1=0.89$) when trained on Wikipedia data. The XLM-R outperformed rule-based and LSTM methods. However, performance was reduced in out-of-domain scenarios, especially for isiZulu legal texts ($F1=0.45$) and organizational entities across all tests (the lowest $F1=0.19$). The models are accurate on Wikipedia texts while having issues with domain adaptation. This particular Wikipedia-based weak-supervision method was used considering its robustness to complexities involving morphologically complex languages and its ease of adaptability.

2.1.1 NER for Sinhala Language

Sinhala, the native language of Sri Lanka, belongs to the Indo-Aryan branch of the Indic language family and has approximately 20 million speakers. Although there are many successful NER systems for the English language, these systems cannot be directly applied to Indic languages due to their linguistic features (J. Dahanayaka and R. Weerasinghe 2014). Considering the lack of resources and inherent different linguistic language features, much of the research on Sinhala NER focused on data- driven and rule-based solutions.

Dahanayaka and Weerasinghe’s (J. K. Dahanayaka and A. R. Weerasinghe 2014) work focuses on identifying named entities in an unstructured text. This work doesn’t focus on categorizing the identified named entities. The researchers assume the methods employed for Indic languages will probably work on Sinhala as well. Consequently, Conditional Random Fields (CRF) and Maximum Entropy (ME) were chosen as methods for experiments. For both methods, appropriate feature sets are refined. For ME, after several experiments, Context word (window size = 1) and language-dependent features (preceding word/words) are selected. Under CRF context-word, word, suffix, and bi-gram features are selected as feature sets. The results highlight that CRF outperforms ME in Sin-

Sinhala NER and the applicability of statistical methods for Sinhala language NER with a high precision rate of 91.64 and recall of 69.34.

Udeshika and Attanayaka (Senevirathne et al. 2015) build upon Dahanayaka (J. Dahanayaka and R. Weerasinghe 2014) research by not only detecting but also categorizing the entities. The study utilizes the CRF method, which is based on previous work. It incorporates language-inherited features such as word-level features: suffix and prefix information, first word, word length, and preceding tag. The optimal window length is chosen to capture context effectively, and the best feature set is selected for each class. A window size of 5 was identified as the best optimal window size for both prefix and suffix features. For context, word feature two different window sizes showed the best results for different classes. In this approach, they are considered four named tags as person (NEP), location (NEL), organization (NEO), and not named entity as NEN. Considering several feature combinations, it trained separate models and observed the highest F-value for NEP tag as 73.46, NEO as 66.78, and NEN as 71.73 in the last few feature combinations, but NEL gives it the highest value of 60.05 in the 3rd feature combination.

The Ananya (ananya) (Manamini et al. 2016) system initially classified entities into only three categories in Sinhala. The researchers proposed enhancing the previous UCSC binary corpus for NER use by (J. Dahanayaka and R. Weerasinghe 2014) study by adding more context through online newspaper articles to expand the corpus to 110,000 words. Different models are trained under ME and CRF using different feature combinations such as context words, prefixes and suffixes, word length, word frequency, first and last word in a sentence, gazetteer list, POS tags, etc. They have mainly considered the works of (ibid.) and (Saha and Ekbal 2013) when identifying the candidate set of features for the experiments. The combination that gave the maximum F1 score can be identified as follows: clue words, context words (window size = 1), n-gram feature (n-gram size = 10), gazetteer features, start-end word, and word length feature with a cutoff value of

5. Finally, they concluded that the CRF outperformed the ME method for Indic languages like Sinhala.

The study by Azzez and Ranathunga (Azzez and Surangika Ranathunga 2020b) presents a fine-grained NER-tagged dataset with a trained CRF model and obtained an F1 score of 84.8. Considering the lack of granularity and details in the SOTR Sinhala NER datasets, the researchers have declared a NER tag set according to modern standards with 23 categories (1.0 12 OntoNotes (Linguistic Data Consortium 2007) was refined for this purpose). For annotation, human annotators and the Inception tool were used to annotate the documented 70k token corpus. For future work, it suggests a sizeable, fine-grained dataset (300k tokens) so that deep learning can be used.

Another study discusses a rule-based solution for Sinhala NER, especially on a sports dataset. Initially, sport-related data was collected with the help of Sinhala e-sport articles, and class labels were automatically identified by using a rule-based, semi-automated mechanism. After that, those identified labels were manually checked with the support of a domain expert prior to the training process. Then annotated data are used to train for Sinhala named entity recognition in the sport domain through multiple models, such as Linear Perceptron, Stochastic Gradient Descent (SGD), Multinomial Naive Bayes (MNB), and Passive Aggressive classifiers. The rule-based solution can be justified due to the domain-specific nature of the question. A class label suggester is basically a semi-automated, rule-based component that is used to annotate the e-sports article and then annotate the data it used to train an NER model through multiple methods. Of these, the multinomial Bayes classifier shows the highest accuracy of 97% in detecting ground, tournament, and other categories. In addition to that, it shows more than 80% precision for the Ground and Other categories (Wijesinghe and Tissera 2022).

Another research successfully improved Sinhala named entity data with a focus on person and location entities using a semi-supervised bootstrapping method.

Increasing Sinhala named entity data using less effort compared to supervised learning is the main intention of the study(Jayasinghe 2017).

2.2 Transfer Learning for low resource languages

Transfer learning is the process of transferring knowledge gained from one or more source tasks to a particular target task(Pan and Q. Yang 2010). In the context of this entity recognition research, transfer learning is instantiated by applying multilingual pretrained source NER models to improve the performance on the target Sinhala NER task. Transfer learning is mainly useful with the prediction of low-resource problems where we are unable to extract more data due to their linguistic complexities. In that situation, we can predict the behavior of such a low-resource problem with the help from previously trained or learned model knowledge. In this case we can abstract knowledge to pretrained transformer embeddings. Figure 2.1 shows the illustration of the learning processes of traditional machine learning (ML) and transfer learning (TL).

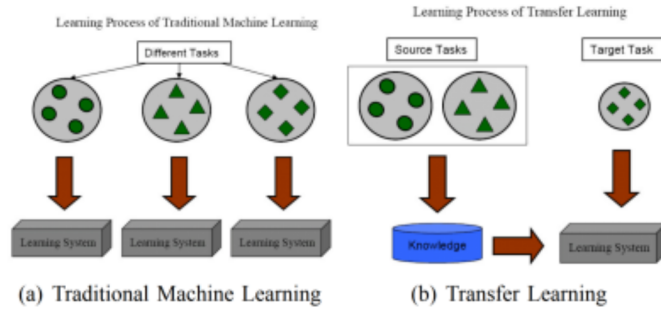


Figure 2.1: Traditional Machine Learning vs Transfer Learning (Pan and Q. Yang 2010)

Hence, with the availability of pretrained weights, through transfer learning we can reuse the existing source model weights for the prediction on low-resource target models without worrying about less labeled data. Compared to the traditional machine learning models, here we are not going to worry about the number of labeled data available because of the knowledge transfer technique. This removes the need for developing models from scratch, as pretrained models can be

fine-tuned on task-specific data or tasks upon requirement.

Now transfer learning can be used for task-specific purposes or feature extraction purposes. For example we can use **indicNER**, which is an NER model trained on Indic languages that can be directly used or fine-tuned on Sinhala NER data as preferred. But we can also use **IndicBERT** to extract embeddings, which can be later used for tasks like Sinhala NER. Lets focus on some of the previous work on using transfer learning on NER.

In Indic language NLP research, it has been shown that multilingual fine-tuning efficiently leverages language relatedness, leading to improvements over monolingual approaches. The Indo-Aryan (IA) language family is chosen for the study, which includes languages such as the exact languages being Bengali, Gujarati, Hindi, Marathi, Oriya, Punjabi, and Urdu. Language relatedness is approximated by the correlation between different languages because similar languages share almost similar linguistic features such as grammar, vocabulary, etymology, and writing systems. The Oriya and Punjabi languages have shown a significant improvement in NER tasks compared to monolingual models. Now the reason for such improvement can be explained because of the low-resource nature of this Oriya and Punjabi languages and they share similar roots when it comes to scripts with languages like high-resource languages like Hindi, whereas Urdu has a completely different script.

As shown in Figure 2.2, incorporating Gujarati training data for Oriya NER yielded a dramatic **54 percentage point improvement** (from 38.8% to 92.4%) in multilingual fine-tuning. Punjabi showed a smaller but notable **6 percentage point gain** when Bengali data was added.

These experiments used **MuRIL** in a study (Khanuja et al. 2021) as the multilingual language model, demonstrating its effectiveness for cross-lingual transfer within the Indo-Aryan language family. Crucially, this work confirms that **strategic language pairing** (e.g., Gujarati→Oriya, Bengali→Punjabi) drives improvements, while **arbitrary data accumulation** provides diminishing re-

turns in Dhamecha et al. 2021.

Train Set	Size	F-Score	Train Set	Size	F-Score
or	1078	0.3882	pa	1409	0.8535
Base set: or			Base set: pa		
+ gu	3425	0.9245	+ bn	21579	0.9156
+ bn	21379	0.8836	+ mr	13795	0.8883
+ hi	10590	0.8795	+ hi	10970	0.8759
+ pa	2487	0.8657	+ gu	3805	0.8673
+ mr	13415	0.8649	+ or	2487	0.8426
Base set: or+gu			Base set: pa+bn		
+ bn	23725	0.9301	+ hi	31270	0.9286
+ pa	4884	0.9231	+ mr	34095	0.9160
+ hi	12936	0.8836	+ or	22838	0.9137
+ mr	15761	0.8855	+ gu	24105	0.9105
Base set: or+gu+bn			Base set: pa+bn+hi		
+ mr	36061	0.9151	+ mr	43606	0.9211
+ pa	25184	0.9036	+ or	32349	0.9156
+ hi	33236	0.8916	+ gu	33616	0.9132
Base Set: or+gu+bn+mr			Base set: pa+bn+hi+mr		
+ hi	45572	0.9419	+ gu	45952	0.9567
+ pa	37520	0.8922	+ or	44685	0.9231
All	46665	0.8848	All	46665	0.9086

Figure 2.2: Study of graded addition of languages for NER task on low resource languages of(left) Oriya and(right) Punjabi using MuRIL(Dhamecha et al. 2021)

Another study establishes a baseline for NER for low-resource languages like Upper Sorbian and Kashubian through West Slavic language family (Polish and Czech) feature extractions. This study targets three RoBERTa models that were built from scratch, including two mono-lingual Polish and Czech models and one bi-lingual model for that same languages. These models were evaluated on the entity recognition task for Czech, Polish, Upper Sorbian, and Kashubian, while comparing the available models like RobeCzech, HerBERT, and XLMR.

Through their findings, they suggest that mono-lingual models perform well with their languages, and both monolingual and West Slavic language family models identified named entities better than larger multi-lingual models. This research highlights the efficacy of leveraging shared linguistic features and larger datasets within the same language family. Although we cannot definitively conclude the superiority of multilingual fine-tuning, it is observed that both family-

specific and monolingual-trained models outperform large multilingual models (Torge et al. 2023).

Under this study, transfer learning was used to leverage data from source languages to Galician , West Frisian , Ukrainian, Marathi, and Tagalog. All the target languages outperform log-linear CRF in low-resource settings. A BiLSTM+CRF with character-level CNN is used as the neural network. After adding a source language such as Spanish to the target language Galician , F1 increases to 76.40 for the neural CRF and 71.4 for the log-linear CRF. The trend is similar for other source languages, such as Catalan (75.40) and Italian (70.93) (Cotterell and Duh 2017).

In another study , models trained with multilingual data outperform models trained with individual datasets when tested on languages. The authors focus on using Marathi and Hindi languages as assisted languages for each language. The paper also emphasizes the inefficacy of arbitrary addition of datasets in NER. The observations portray that XLM-Roberta, RoBERTa Hindi, MahaBERT , and MahaRoBERTa perform better on the mixed dataset than on the monolingual IIT Bombay dataset (Marathi). Again, this demonstrates that models trained with diverse, multilingual data are able to leverage cross-linguistic features and knowledge, leading to better performance than those trained on single-language datasets (Sabane et al. 2023).

Another study focuses on three different transfer learning architectures, including cross-domain transfer, cross-application transfer, and cross-lingual transfer. Under cross-domain transfer, it learns target domain entity tagging from the knowledge extracted from the source domain. When it comes to cross-application transfer, it assumes that knowledge will be extracted from different applications using the same alphabet to make the prediction on target languages by simply altering the CRF layer. Cross-lingual transfer relates to additional multilingual languages, which is sensitive to size and the quality of language selection. Through this study, the researchers focused on transferring knowledge between languages

that have similar alphabets, such as English and Spanish, due to the difficulty of transferring knowledge between two distinct languages. That’s why we need to focus on language similarities when we are considering multilingual model-level transfer learning.

In cross-lingual transfer learning, all layers of the pretrained models are transferred through exploiting the morphologies shared by the two similar languages. For the transfer learning model, a hierarchical framework is proposed. This framework involves extracting detailed character-level features, combining them with word-level features and context, and then predicting the sequence of labels with the CRF layer.

The transfer learning model used hyperparameters as character embedding at 25, word embedding dimension 50 for English and 64 for Spanish, and an initial learning rate of 0.01. In Figure 2.3, the results for the Spanish and Dutch NER achieve state-of-the-art, the highest entity recognition performance among all the benchmark datasets that they have considered. Also, they have concluded that target language label abundance, correlation between source and target, and parameter selection have an impact on the transfer learning model predictions in the NER domain(Z. Yang, Salakhutdinov, and W. W. Cohen 2017).

Model	CoNLL 2000	CoNLL 2003	Spanish	Dutch	PTB 2003
Collobert et al. (2011)	94.32	89.59	–	–	97.29
Passos et al. (2014)	–	90.90	–	–	–
Luo et al. (2015)	–	91.2	–	–	–
Huang et al. (2015)	94.46	90.10	–	–	97.55
Gillick et al. (2015)	–	86.50	82.95	82.84	–
Ling et al. (2015)	–	–	–	–	97.78
Lample et al. (2016)	–	90.94	85.75	81.74	–
Ma & Hovy (2016)	–	91.21	–	–	97.55
Ours w/o transfer	94.66	91.20	84.69	85.00	97.55
Ours w/ transfer	95.41	91.26	85.77	85.19	97.55

Figure 2.3: state of the art results for Spanish and Dutch NER (Z. Yang, Salakhutdinov, and W. W. Cohen 2017)

Now let’s analyze a more comprehensive research work done by Murthy and Bhattacharyya (Murthy and Bhattacharyya 2018) in 2018 on improving NER tagging performance in low-resource languages via multilingual learning. Multilingual learning in this context refers to where a deep neural network is trained

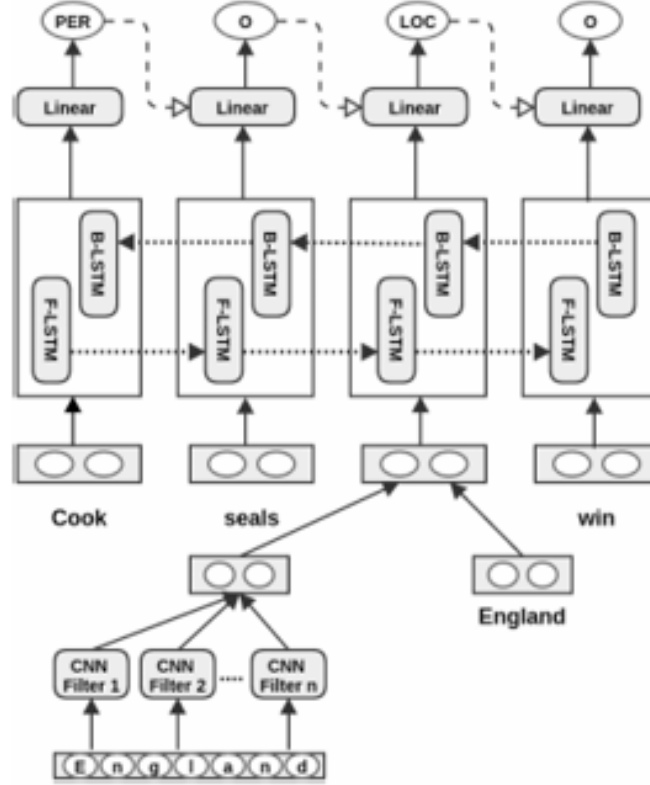


Figure 2.4: CNN Bi-LSTM model for NER used by (Murthy and Bhattacharyya 2018).

for the same task in multiple languages by sharing some or all layers of the neural network between related languages. Accordingly, for low-resource language setup, Marathi, Bengali, Tamil, and Malayalam as the primary languages and Hindi as the assisting language have been considered.

The above languages are all related in the sense that they are all members of the Indo-Aryan language family. Indo-Aryan languages in the Indian subcontinent share the same set of phonemes, and the correspondence between characters across scripts can be easily established (Subbarao 2012). In the monolingual learning setup, the CNN Bi-LSTM model (see Figure 2.4) and CRF model with traditional features are employed in the research. For the multilingual learning setup, CNN Bi-LSTM Sub-word (figure 2.5) and CNN Bi-LSTM All (all layers are shared) are used .

In the monolingual learning setup, the CNN Bi-LSTM model outperforms

Approach	Tamil	Malayalam	Bengali	Marathi
CRF + POS	44.60	48.70	52.44	44.94
CNN Bi-LSTM	52.34	55.37	50.34	56.53
CNN Bi-LSTM + Subword	52.34	56.82	52.56	50.25
CNN Bi-LSTM All	53.47	56.75	53.90	57.37

Figure 2.5: Comparison of Various Multilingual Learning Strategies and Monolingual Deep Learning Systems on Tamil, Malayalam, Bengali, and Marathi NER Using Hindi as Assisting Language

the CRF system on three out of four languages. The results from the Table 2.5 show that the CNN Bi-LSTM All model significantly outperforms both the CNN Bi-LSTM and CRF models for NER across four Indian languages: Tamil, Malayalam, Bengali, and Marathi. The CNN Bi-LSTM all model surpasses the CNN Bi-LSTM sub-word model in three of these languages, with only a slight advantage of the sub-word model in Malayalam. This may be due to the rich morphology in Malayalam, which helps the sub-word model to capture the meaning of words better. Incorporating Hindi as an assisting language further enhances NER performance for all languages, with the CNN Bi-LSTM all model achieving an increase in F-score of at least 0.8 absolute points over the baseline monolingual systems.

Chapter 3

Design

The primary objective of this research is to investigate the applicability of transfer learning for the Sinhala Named Entity Recognition (NER) task. In this context, target domain predictions is based on the knowledge extracted from the pre-trained source model. Here, first it is necessary to select the appropriate source model, which is trained on a large corpus, and then fine-tune it for the Sinhala NER task prior to knowledge transfer.

3.1 IndicBERT

The language modeling has evolved from static embeddings such as word2vec to sequence aware models such as LSTM and RNN. Later, in 2015, the attention technique, which targeted the highest priority part of the given input text, was introduced, and as a result of that, transformers and BERT models were introduced. Although all the existing models before the transformers did not consider the global context. The usage of words changes according to the context, and because of that, it reduces the accuracy in NER tasks. Through transformers and BERT, we can handle that issue.

The BERT (Bidirectional Encoder Representations from Transformers)(Devlin et al. 2019) model and all its the derivatives are based on the Transformer Architecture(Vaswani et al. 2017). In a transformer there is a decoder stack and an encoder stack , each of which must contain one attention layer at the bottom.

Each encoder can be divided into two sub parts: self-attention and feedforward. The self-attention mechanism understands how words relate to each other and passes that along to the upper layers. The feedforward layer will output

the self-attention layer and feed it to the feedforward neural network for further analysis. The encoder enhances the representation of words in all its layers. The decoder has a self-attention layer followed with an encoder-decoder attention layer and a feedforward layer. This kind of full encoder-decoder setting is used for sequence-sequence tasks like machine translation. This setup allows us to model syntactic/semantic relations between words and reduce feature engineering. In the BERT base model 12 encoder layers are employed. We can add more layers on top of BERT for specific tasks.

The proposed architecture for the system consists of 3 layers. As per it shown in Figure 3.1. In the first layer, we fine-tuned IndicBERT for the Sinhala Named Entity Recognition task. And the intermediate layer is a linear encoder layer, which is used as a lightweight alternative to the LSTM layers. This linear encoder reshape the contextual embedding output received from the pretrained model to the final CRF layer.

The choice of **IndicBERT** (Doddapaneni et al. 2023) depends on the main reasons for its invention. IndicBERT is a multilingual model that is trained on 12 Indic languages. This model was designed to cover Indic languages as much as possible with a high model capacity covering related languages. Previous work shows that multilingual models trained using pretraining data from a smaller set of related languages lead to better performance on downstream tasks than large-scale models that support many languages (Conneau et al. 2020; Khanuja et al. 2021). In this research, IndicBERT is fine-tuned with Sinhala and Bengali data.

3.2 CRF

Transformer models like **BERT** and **RoBERTa** model the probability distribution of potential tags for each token in a sequence independently. But this method ignores the interdependent relationship between tags. For example, you can't have **B-LOC** (beginning of a location) followed by **I-PER** (inside of a person).

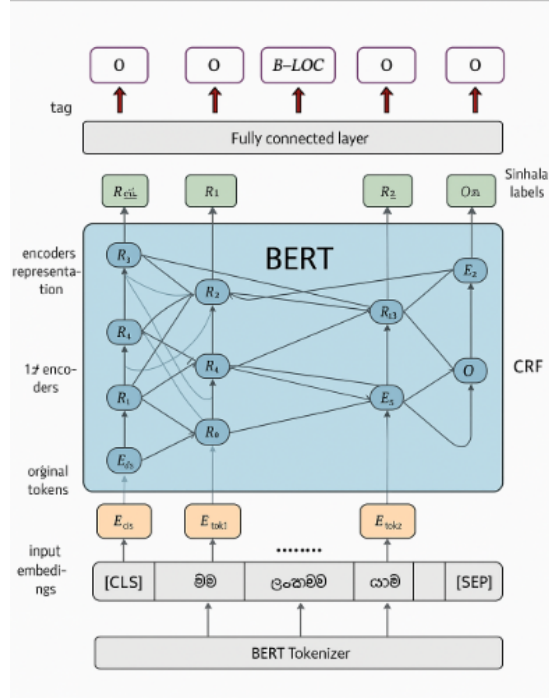


Figure 3.1: A layout of the **BERT-CRF** model.

So to model tagging of tokens jointly, we use the CRF (Conditional Random Fields) method (Lafferty, McCallum, and Pereira 2001). It is an alternative to Hidden Markov Models (HMM) and is still going to be used to incorporate the sequential structure. When considering entity recognition, HMM does not work properly due to its limitations in contextual understanding, neglecting rich linguistic features such as POS taggers, and failing to accommodate the ambiguity problem.

In CRF, the transition matrix and emission matrix are considered as key components for building relationships among states and observations. The transition matrix will define the probabilities of transitioning from one state to another, which helps the NER model to understand the sequence of labels or words that appear within the sentence. On the other hand, the emission matrix shows the probabilities of observing or capturing a particular feature, such as word features, contextual features, POS tags, and character-level features, in a given state. Combining these two matrices together helps CRF to effectively identify the dependencies in an accurate manner, which ultimately helps with entity

recognition. Neural networks generate the emission scores automatically without depending on manually defined features.

In CRF, it is necessary to evaluate how well a given label sequence aligns with the observed data, considering the features that we have defined in the model. We are trying to maximize the score value for the correct sequence label while minimizing it for the incorrect labels. Hence, for a prediction sequence $y = (y_1, y_2, \dots, y_n)$ given its input, $X = (x_1, x_2, \dots, x_n)$ we define a score as follows.

$$s(\mathbf{X}, \mathbf{y}) = \sum_{i=0}^n A_{y_i, y_{i+1}} + \sum_{i=1}^n P_{i, y_i} \quad (3.1)$$

In the above equation 3.1, \mathbf{A} represents the transition matrix, where each element $A_{y_i, y_{i+1}}$ is the score for transitioning from tag y_i to tag y_{i+1} . \mathbf{P} is the emission matrix, where P_{i, y_i} represents the score of observing the word x_i being assigned the tag y_i . This matrix P relates to the output of the linear encoder or an LSTM layer.

Now to take the probability to for the whole sequence \mathbf{Y} , we softmax over all possible tag sequences.

$$p(y \mid X) = \frac{\exp(s(X, y))}{\sum_{y' \in \mathcal{Y}_X} \exp(s(X, y'))} \quad (3.2)$$

The denominator of the above function 3.2 is sometimes named as the partition function $Z(x)$ (Al-Qurishi and Souissi 2021).

$$Z(X) = \sum_{y'_1, y'_2, \dots, y'_k} \exp \left(\sum_{i=1}^k E(x_i, y'_i) + \sum_{i=1}^{k-1} V(y'_i, y'_{i+1}) \right) \quad (2)$$

The above partition function is computationally intractable; hence, the forward-backward algorithm is used to avoid checking every possible sequence of tags through brute force. During the training phase, for every classification problem, there should be a loss function to enable model learning. In this context, CRF,

we try to model the probability of a label sequence given an input sentence. So in training we try to maximize this probability to give the correct sequence. But this process involves a lot of calculation, and it doesn't suit a loss function. In consideration of the need for a loss function, negative log likelihood is defined as a parameter that can minimize an objective function .

$$L = -\log (P(y \mid X)) \quad (3.3)$$

$$-\log (P(y \mid X)) = -\log \left(\frac{\exp \left(\sum_{i=1}^k E(x_i, y_i) + \sum_{i=1}^{k-1} V(y_i, y_{i+1}) \right)}{Z(X)} \right) \quad (3.4)$$

$$= \log (Z(X)) - \log \left(\exp \left(\sum_{i=1}^k E(x_i, y_i) + \sum_{i=1}^{k-1} V(y_i, y_{i+1}) \right) \right) \quad (3.5)$$

$$= \log (Z(X)) - \left(\sum_{i=1}^k E(x_i, y_i) + \sum_{i=1}^{k-1} V(y_i, y_{i+1}) \right) \quad (3.6)$$

Chapter 4

Implementation

In this chapter, the methodology and implementation that have been followed during the research are described, and each main section represents how we developed in a descriptive manner. Initially, the appropriate Sinhala dataset is extracted from Wikipedia from scratch, as described in Section 4.1. In Section 4.2, the implementation of the baseline Named Entity Recognition model on CRF for Sinhala is explained to identify the entities in the Sinhala language. Later, in Section 4.3, the fine-tuning process of the existing Sinhala CRF model for entity recognition using a transfer learning approach, especially in the Sinhala language, is explained, including hyperparameter tuning and augmentation approaches. In Section 4.4, the multilingual transfer learning approach from the baseline IndicBERT model, including hyperparameter tuning and augmentation, is mentioned.

4.1 Dataset Preparation

To train our own NER model, it is necessary to have a proper entity tag set in the desired Sinhala language. Data collection or data preparation is considered a crucial step in any machine learning model, and overall model accuracy strictly depends on the quality of the selected dataset. Therefore, it is necessary to have a proper dataset that covers most common Sinhala entity tags; however, the available dataset only covers three categories, such as person, location, and organization entities. As a result of that, we have to extract dataset additionally, considering three categories such as medical, products, and creative work.

There is a study that was conducted by Aravind Krishnan to extract entity

datasets for person, organization, location, and other categories using a weakly supervised approach that collects the training set from Wikipedia (Krishnan et al. 2021). In our research, we have used the same approach as for Sinhala entity recognition from Wikipedia articles, considering person (PER), location (LOC), organization (GRP), products (PROD), medical (MED), and creative work (CW) categories.

Although Sinhala is known as a low-resource, morphologically rich language, the presence of Sinhala Wikipedia articles on the internet has been growing in recent years. Due to language complexities most of NER studies have been carried out by using three major categories as mentioned in the work of (Manamini et al. 2016). That dataset is annotated by IOB format, and they have used the **UCSC** tag corpus (J. Dahanayaka and R. Weerasinghe 2014), which was extended with the contextual data from newspaper articles. But during our research, we are trying to find the applicability of entity recognition by considering more groupings such as person, location, medical domain, creative works, etc.

Since it is hard to obtain entity data belonging to many categories in the Sinhala language, we have to build the dataset for this research from scratch. For bootstrapping training data, we have used the Sinhala Wikipedia articles, and Google Knowledge Graph has been applied to recognize the aforementioned entity types. There are a few reasons for choosing Wikipedia articles over other internet resources, as we can easily find many articles in Sinhala that were created by different communities. This helps to extract different domain features that help with bootstrapping, and we can also easily find different entities due to its vast domain coverage. Our dataset creation contains 4 sub-stages to get the named entities from Sinhala data, as shown in Figure 4.1. In the first stage, it will extract the list of titles from the Wikipedia in the Sinhala language and use the Wikipedia language links to find their English equivalents. After that, it will extract the candidates for named entity tags using Google Knowledge Graphs. Finally, we have used the title lists to annotate Sinhala Wikipedia articles.

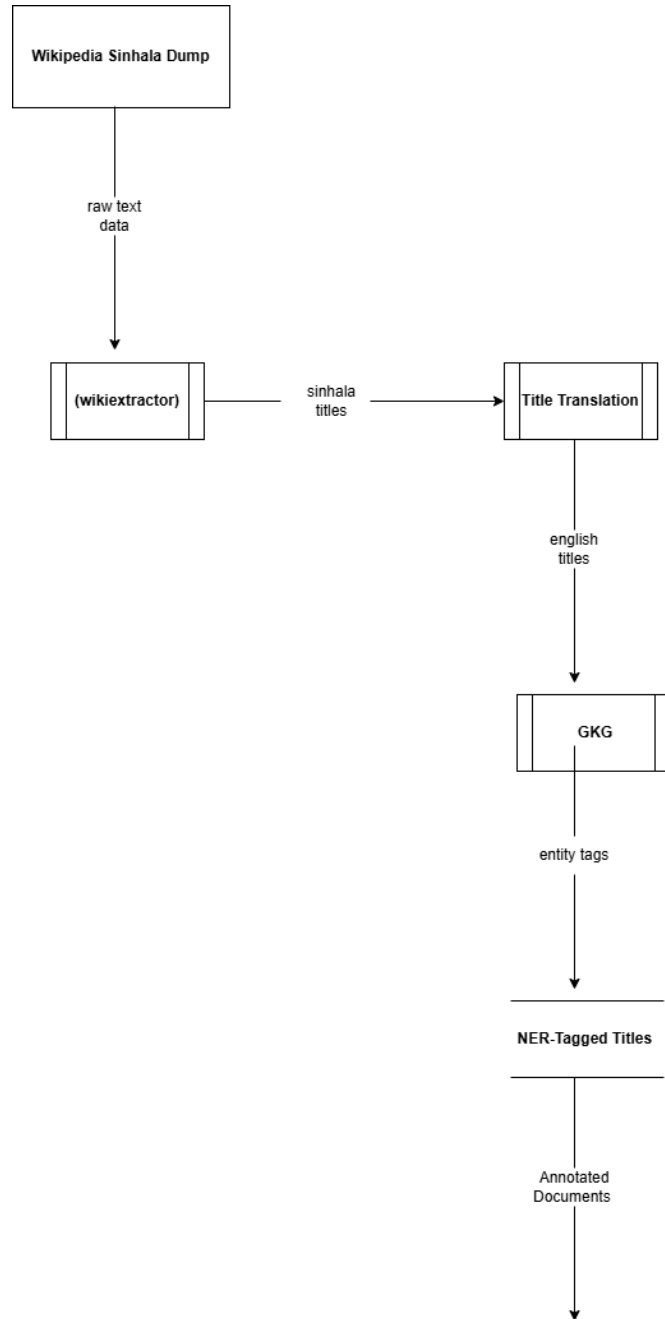


Figure 4.1: Data flow diagram of the data set generation process(Krishnan et al. 2021)

In the first stage of our dataset creation, we extracted the list of article titles from the Sinhala Wikipedia dump and applied preprocessing to remove all the entries that were entirely in different language numbers and characters apart from Sinhala. Also, all the duplicate titles were removed to handle the redundancy problem. Those extracted titles share the primary token and its descriptors within brackets to distinguish entities from each other. The title extraction and preprocessing setup was mentioned below.

```

1  import os
2  import json
3  import argparse
4  import re
5  def contains_sinhala(text):
6      """Check if the text contains at least one Sinhala character."""
7      sinhala_regex = re.compile(r'[\u0D80-\u0DFF]')
8      return sinhala_regex.search(text) is not None
9  def decode_and_filter_titles(input_file, output_file):
10     """Decode titles from JSON file, filter, and save unique Sinhala titles to
        the output file."""
11     seen_titles = set() # Track unique titles
12     with open(input_file, 'r', encoding='utf-8') as infile, open(output_file,
        'w', encoding='utf-8') as outfile:
13         for line in infile:
14             try:
15                 data = json.loads(line) # Parse JSON line
16                 title = data.get('title', '')
17                 # Check if the title is valid, contains Sinhala characters,
                    and is not a duplicate
18                 if title not in seen_titles and contains_sinhala(title):
19                     outfile.write(title + '\n') # Write the title to the
                        output file
20                     seen_titles.add(title) # Mark title as seen
21             except json.JSONDecodeError:
22                 continue # Skip invalid JSON lines

```

After that, we queried each title in the previously extracted and preprocessed title list to allocate their respective named entity tags using Google Knowledge Graphs, which are similar to the tags used in NER. During our study, the tags were restricted to PER, GRP, LOC, MED, PROD, and CW. Entities that do

not belong to the above-mentioned tags or groups are considered as OTHER. In Google Knowledge Graphs, queries are only accepted in English; therefore, it is necessary to translate the extracted queries from Sinhala into English. They use different sources when generating tags, and they generate a list of tags depending on their rankings.

Later, we selected the top one. After that, if one of those tags appeared in the extracted title list, it is assigned, and if not, it will be annotated as the tag OTHER. Finally, we automatically annotate the text of each Wikipedia article in Sinhala by considering our selected six tags, and how it does so is illustrated in the below Appendix A.1.

Previously, we defined all the categories that we are interested in in Sinhala, and then we had to build the connection to access the Wikipedia articles and fetch contents from them. If our fetch request was successful, then the content was extracted and broken into sentences using NLTK’s sentence tokenizer. Then each sentence is further tokenized into separate words, and each of them passes to the multi-word entity labeling function, which is described below in the Appendix A.2.

Multi-word entity recognition in Sinhala is a crucial part of Named Entity Recognition (NER), and it helps to identify and classify various words that represent specific entities such as person, locations, organizations, creative works, etc. Compared to other languages, Sinhala’s multi-word classification is a bit challenging because of its unique syntax and morphology structure. However, by recognizing these multi-word entities, it will give more contextual understanding of the text, which ultimately helps with accurate entity identification. The following code segment explains how we manage to handle the tokenization and labeling process of the multi-word entities.

Firstly, we applied the tokenization by splitting the given input Sinhala string into individual words, considering space. Next, using the bigrams (two-word phrases) and trigrams (three-word phrases) within the text, it will find a match

from the dictionary, which records all the known entities with their corresponding types, and label them accordingly. During the labeling process, it will record a "B-" tag for the first word, which indicates the beginning of an entity, and an "I-" tag for the subsequent word, which indicates the remaining part of the same entity. The same labeling approach is done as it is except for the "Other" category. Overall, the following code explains how we managed to identify and categorize multi-word entities in Sinhala.

```

1  # Function to tokenize and correctly label multi-word entities
2  def tokenize_with_multiword_entities(text, result):
3      words = text.split() # Simple space-based split
4      i = 0
5      while i < len(words):
6          match_found = False
7          # Check for bigrams (2-word entities) and trigrams (3-word entities)
8          for n in [3, 2]:
9              if i + n <= len(words):
10                 phrase = " ".join(words[i:i+n]) # Create an n-word phrase
11                 if phrase in result:
12                     entity_type = result[phrase]
13                     tokens = phrase.split()
14                     if entity_type == "Other":
15                         yield from ((tok, "Other") for tok in phrase.split())
16                     else:
17                         yield (tokens[0], f"B-{entity_type}") # First token ->
18                         # B-tag
19                         for tok in tokens[1:]:
20                             yield (tok, f"I-{entity_type}") # Remaining tokens ->
21                             # I-tag
22                         i += n # Move ahead by n words
23                         match_found = True
24                         break
25             if not match_found:
26                 word = words[i]
27                 if word in result:
28                     entity_type = result[word]
29                     if entity_type == "Other":
30                         yield (word, "Other") # Not an entity
31                     else:
32                         yield (word, f"B-{entity_type}") # Single-word entity
33                 else:
34                     yield (word, "Other") # Not an entity
35                 i += 1 # Move to next word

```

When it comes to entity recognition, it is necessary to identify the ambiguous titles in a much more accurate manner, and for that, we have used hyperlinks. Then all existing hyperlinks were annotated with their tags.

CRF models will rely on part-of-speech tagging (POS) features to make their prediction on a given NLP task. But when we consider other neural network mod-

els, they will automatically learn their linguistic features from the given dataset, whereas CRF needs explicit feature analysis to capture linguistic patterns. We used the POS tagger introduced by (Fernando and S. Ranathunga 2018), along with the tag set developed by (Fernando, S. Ranathunga, et al. 2016), for the POS tagging task. This Sinhala POS tagger is a fine-tuned version of the original TnT POS tagger introduced by (Brants 2000). The parameters relevant to Sinhala were already generated by (Fernando and S. Ranathunga 2018).

The provided POS tagger was installed, and then the environmental variables were set as follows.

```
nvim ~/.bashrc

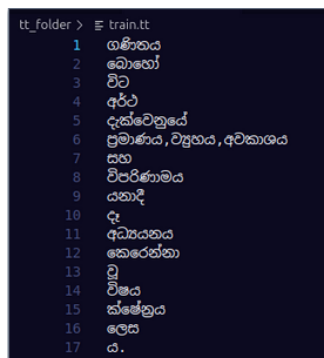
export PATH=$PATH:/home/kavisha/Research/tnt/

export TNT_MODELS=/home/kavisha/Research/tnt/models

source ~/.bashrc
```

Then the input file was in the format **tt** and it was tagged using the provided parameters for Sinhala (see Figure 4.2). The model parameters were in 2 separate files called `sinhalafinal.lex` and `sinhalafinal.123` as shown in Figure 4.3. Then we can use the generated POS tag files in Figure 4.5 for feature engineering purposes in CRF model training by combining them with usual NER tagged data files.

```
tnt sinhala_final train.tt > train.tts
```



```
tt_folder > train.tt
1 ගම්මානය
2 බෝහෝ
3 විට
4 අර්ථ
5 දැක්වෙනුයේ
6 ප්‍රමාණය, විස්තරය, අවකාශය
7 සහ
8 විපරිණාමය
9 යනාදී
10 දෑ
11 අධ්‍යයනය
12 කෙරෙන්නා
13 වූ
14 විෂය
15 ක්ෂේත්‍රය
16 ලෙස
17 ය.
```

Figure 4.2: Input file format for the POS tagger.

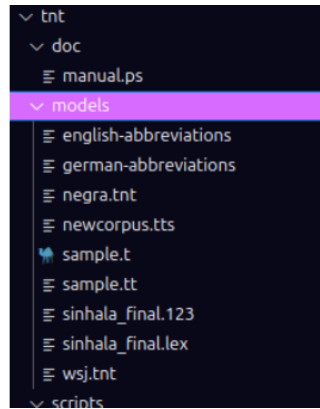


Figure 4.3: Model files for the Sinhala POS tagging

```

tt_with_pos_tags > F train.tts
1  %% tnt statistically tagged file, Wed Apr 2 14:29:31 2025
2  %% lexicon : /home/kavisha/Research/tnt/models/sinhala_final.lex
3  %% ngrams : /home/kavisha/Research/tnt/models/sinhala_final.123
4  %% corpus : ../tt folder/train.tt
5  %% model : trigrams
6  %% sparse data : linear interpolation
7  %% lambda1 = 2.388816e-01 lambda2 = 2.858568e-01 lambda3 = 4.752616e-01
8  %% unknown mode: statistics of singletons
9  %% case of characters is significant
10 %% using suffix trie up to length 10
11 %% suffix backoff with theta = 7.186713e-02
12 %% Thorsten Brants, thorsten@brants.net
13 ගම්මාන NNC
14 බොහෝ RP
15 එම POST
16 අර්ථ JCV
17 දැක්වෙනුයේ VP
18 ප්‍රමාණය, එහෙය, අවසානය NNC
19 සහ CC
20 විද්වත්වීම NNC
21 යනාදී POST
22 දෑ NNC
23 අධ්‍යයනය NCV
24 කෙරෙන්නා VP
25 මා VP
26 විෂය NNJ
27 සංකල්ප NNC
28 ලෙස POST
29 ය. ???
30
31 ගම්මානගෙන් NNC
32 බොහෝ RP
33 දෙකෙකු NNC
34 විසින් POST
35 දරන VP
36 තරින් DET
37 මතකයේ NNC
38 නමුත් NNC
39 ප්‍රකාශන NNC
40 සහ CC
41 අර්ථදැක්වීමටමත් NNC
42 ඉංග්‍රී VNF
43 නිගමන JJ
44 තර්කනය NNC
45 මගින් POST
46 සතර RPCV
47 කළ VP
48

```

Figure 4.4: POS tagged tokens in .tts format

4.2 Implementation of the Baseline NER model

In transfer learning, we interact with two models known as the source and target models for a given task. The source model is trained with a large dataset, and based on the knowledge it obtained during the entity recognition task, it will do the prediction on the same task in the target model, which we interact with in low-resource languages that are similar to the source model. This research is focused on reviewing the applicability of transfer learning for the Sinhala language by considering both monolingual and multilingual approaches.

The Sinhala baseline NER model was implemented using the work carried out by a group of researchers at the University of Moratuwa in 2020 (Azeez and Surangika Ranathunga 2020b). The NER task is considered a sequential classification problem, and most statistical models, such as Hidden Markov Models (HMM), Maximum Entropy Models (ME), and Conditional Random Fields (CRF), have been used for this purpose. Among these, CRF has performed well in most languages. Because of that, we also used the CRF model as our baseline model in the monolingual approach. However, there is no readily available NER model for the Sinhala language; therefore, based on the explanation in the work of Azzes, we had to implement the model from scratch.

4.2.1 Splitting the NER dataset

In every machine learning model, the dataset has to be split into three parts: a training set, a testing set, and a validation set. For the training of the monolingual baseline NER model, we have used the Ananya dataset, which contains 10,000 annotated sentences under four entity tags, such as person, organization, location, and other (Manamini et al. 2016). They have used the automated and manual annotations for their dataset creation. In the training, testing, and validation, we have used 2668094, 1222781, and 1222781 tag entities in the Sinhala language, respectively, and their distribution is mentioned in the table 4.1 below. Here, each train, dev, and test dataset covered 11 categories, such as 'B-CW', 'B-GRP', 'B-

LOC', 'B-PER', 'B-PROD', 'I-CW', 'I-GRP', 'I-LOC', 'I-PER', 'I-PROD', and 'Other'.

Category	Frequency	
	<i>Train</i>	<i>Dev and Test</i>
Other	2634792	1201619
B-PER	37115	3651
I-PER	2466	1882
B-CW	3113	8971
B-LOC	15612	1703
I-LOC	2681	2084
I-CW	650	1500
B-GRP	3716	567
I-GRP	1262	737
B-PROD	85	48
I-PROD	2	19

Table 4.1: Distribution of train, dev and test dataset

4.2.2 Implementation of the monolingual CRF model

Bi-LSTM (Bi-Directional Long Short-Term Memory) together with the CRF layer is considered the most effective approach in deep learning. However, they work well with the large dataset. In our research for monolingual baseline experiments, we only have 10,000 sentences of annotated data; therefore, we had to use the CRF model instead of BiLSTM-CRF.

We used the PyCRFSuite¹ to implement the PyCRF model. The PyCRF model can effectively identify the relationship among adjacent labels or tags, which is crucial for identifying the accurate entity tags in Sinhala. In addition to that, due to its flexibility, it allows us to incorporate various features, which helps with the NER task. Feature extraction is necessary for entity recognition in any language with a CRF model because it provides an ability to make accurate predictions on entities by referring to the context of a given sentence. In this research, manually designed features, such as word length, word position, prefix and suffix, POS tags, endings or beginnings, etc., were extracted as tokens, and it is mentioned in the below Appendix A.3.

¹<https://github.com/scrapinghub/python-crfsuite>

```

# Define the entity categories as lists
location_entities = [
    "ආසියානු", "ශ්‍රී ලංකා", "ඉන්දිය", "ඉන්දියාව", "රාජ්කොට්", "සෞරාෂ්ට්‍ර",
    "සසෙක්ස්", "වේල්ස්හි", "කෙන්සිංටන් මාද්‍රිගාව", "බකිංහැම් මාද්‍රිගාව"
]

person_entities = [
    "හර්දික් පාණ්ඩ්‍යා", "සුර්‍යකුමාර් යාදව්", "ශුබ්මත් ගිල්", "පැකුම් නිශ්ශංක",
    "කුසල් මෙන්ඩිස්", "දසුන් යානක", "අක්සාර් පටෙල්", "විලියම්", "හැරි", "මෙගන්", "චාල්ස්"
]

organization_entities = [
    "ක්‍රිකට් සංගම්", "ගාඩ්‍යන් පුවත්පත"
]

date_entities = [
    "ලබන 10 වැනිදා"
]

event_entities = [
    "ක්‍රිකට්", "20/20 ක්‍රිකට් තරගාවලිය", "20/20 ක්‍රිකට් තරගය", "තරගය", "T20 තරග",
    "තරග", "එක්දින තරගාවලිය", "පළමු තරගය", "රජුගේ රාජාභිෂේකය"
]

```

Figure 4.5: Gazetteer list used in the Research

These features focus on word-level features like length, prefixes, and position. Clue words, which help to differentiate entities with person, location, or organization. Moreover, gazetteer is used to enhance the performance in the NER task, but there are some features that are not very helpful in Sinhala entity recognition. Since Sinhala is a case-insensitive language, features like "word.isupper" won't be that helpful in recognizing patterns.

```

1 # gazetteer.py
2 # Define the entity categories as lists
3 def get_gazetteer():
4     """
5     Returns the entity lists as sets for fast lookup.
6     """
7     return {
8         'locations': set(location_entities),
9         'persons': set(person_entities),
10        'organizations': set(organization_entities),
11        'dates': set(date_entities),
12        'events': set(event_entities),
13    }

```

Then the train, dev, and test datasets were loaded. After that, it will process the previously split training, testing, and validation datasets by cleaning the

text, tokenizing them, and converting them into numerical format for the CRF model to understand. Because machine learning models will only recognize the numerical data, it is necessary to convert it prior to training. Later, save all the processed sentences into three separate files for future reference.

```

1
2 train_data = pd.read_csv("/content/drive/MyDrive/neural_crf/data/base_mode_cmp
   /train.tsv",sep="\t")
3 train_splitter = SentenceSplitter(train_data)
4 val_data = pd.read_csv("/content/drive/MyDrive/neural_crf/data/base_mode_cmp/
   validation.tsv", sep="\t")
5 val_splitter = SentenceSplitter(val_data)
6 test_data = pd.read_csv("/content/drive/MyDrive/neural_crf/data/base_mode_cmp/
   test.tsv",sep="\t")
7 test_splitter = SentenceSplitter(test_data)
8
9 # Prepare all three datasets
10 train_sents = prepare_sentences(train_splitter.split_sentences())
11 val_sents = prepare_sentences(val_splitter.split_sentences())
12 test_sents = prepare_sentences(test_splitter.split_sentences())
13
14 save_sentences_to_file(train_sents, 'train_sentences.txt')
15 save_sentences_to_file(val_sents, 'val_sentences.txt')
16 save_sentences_to_file(test_sents, 'test_sentences.txt')

```

Afterwards the features, such as word length, gazetteer features, suffix, prefix, etc., were extracted from each sentence with the corresponding labels. The following snippet of code related to feature extraction mentioned that for every sentence in train_sentence.txt, we have to extract features that we mentioned above, and it will use them as input to our baseline CRF model. It will do the same for the testing and validation set as well. Unlike neural networks, instead of the raw input, a feature set is fed to the model.

```

1
2 # Feature extraction for all sets
3 X_train = [sent2features(s) for s in train_sents]
4 y_train = [sent2labels(s) for s in train_sents]
5
6 X_val = [sent2features(s) for s in val_sents]
7 y_val = [sent2labels(s) for s in val_sents]
8
9 X_test = [sent2features(s) for s in test_sents]
10 y_test = [sent2labels(s) for s in test_sents]

```

Some small number of hyperparameters were tuned following the feature extraction, as it is shown below in the Appendix A.4

A final model is trained using a previously created preprocessed training dataset and selected hyper-parameters for our dataset as it shows below.

```

1 #Train final model with best parameters
2 final_trainer = pycrfsuite.Trainer(verbose=True)
3 for xseq, yseq in zip(X_train, y_train):
4     final_trainer.append(xseq, yseq)
5 final_trainer.set_params(best_params)
6 final_trainer.train('best_crf_model.crfsuite')

```

Finally, it is evaluated with the testing dataset for the best parameters considering measurements like precision, accuracy, and F1 score, as shown in the below code snippet.

```

1
2 # Evaluate on test set
3 final_tagger = pycrfsuite.Tagger()
4 final_tagger.open('best_crf_model.crfsuite')
5 y_test_pred = [final_tagger.tag(xseq) for xseq in X_test]
6 print("\nFinal Test Performance:")
7 print(classification_report(
8     [label for sent in y_test for label in sent],
9     [label for sent in y_test_pred for label in sent]
10 ))

```

4.3 Implementation of the Monolingual Transfer Learning Model

Transfer learning works well on NLP tasks, especially in low-resource entity recognition tasks. At the beginning, researchers were focused on using pretrained models such as Word2Vec and GloVe for the purpose of creating embeddings on a given task, which were fine-tuned for NER. In 2018, with the introduction of transformer-based models like BERT (Bidirectional Encoder Representations from Transformers), many researchers focused on finding the applicability of NER tasks for resource-limited languages like Sinhala. When it comes to resource-limited settings, transfer learning helps to adapt these powerful models known as source models, which are trained with large data coverage, to predict outcomes in new languages with minimal data coverage. During our research, we exploit the pretrained multilingual model, which is trained for languages using transformer technique, as our source model. Here, our main purpose is to predict Sinhala entity recognition using the knowledge extracted from the multilingual NER model.

As explained under the research design earlier, we use Indic-BERT as the pre-trained model to transfer knowledge to target language predictions, and it is known as the source model. And furthermore, we add a CRF layer on top of the IndicBERT layers to suit our downstream task. Because it will be used to enhance the model’s accuracy for named entity recognition, or NER, tasks. The CRF layer is beneficial for identifying the dependencies between the output labels, which is helpful with accurate prediction of NE tags based on the context of the whole sequence rather than individual tokens.

The architecture was implemented from the flexible BERT-BiLSTM-CRF framework of Allan Jie (Jie 2020). This framework, implemented from PyTorch, contains configurations for an embedding layer, a Bi-LSTM layer, and an inference layer. Early neural architectures of NER employed LSTM layers (Lample et al. 2016) to get rich contextual embeddings, but now large models like BERT

can capture long dependencies more effectively with the self-attention mechanism. Because it will help the model to consider the various words in a sentence relative to each other. Hence the choice was to add a single linear encoder between embedding and inference layers. Through that, we can enhance the model's long dependency capturing abilities in the given input data. Through that, we can easily handle the complex language-like Sinhala named entity recognition task.

The following code illustrates how we apply it with the Indic-BERT model. It initializes the transformer embedder and sets up the encoder based on the hidden dimension mentioned above. If that value is positive, then it uses the BiLSTM encoder; else, it goes with the default linear encoder. Additionally, it is necessary to have a fixed length for the sequence, and if any sentence is shorter than the predefined length, padding is added to make it to the required length.

```

1  import torch
2  import torch.nn as nn
3  from src.model.module.bilstm_encoder import BiLSTMEncoder
4  from src.model.module.linear_crf_inferencer import LinearCRF
5  from src.model.module.linear_encoder import LinearEncoder
6  from src.model.embedder import TransformersEmbedder
7  from typing import Tuple, Union
8  from src.data.data_utils import START_TAG, STOP_TAG, PAD
9  class TransformersCRF(nn.Module):
10     def __init__(self, config):
11         super(TransformersCRF, self).__init__()
12         self.transformer = TransformersEmbedder(transformer_model_name=config.
13             embedder_type)
14         if config.hidden_dim > 0:
15             self.encoder = BiLSTMEncoder(label_size=config.label_size,
16                 input_dim=self.transformer.get_output_dim(),
17                 hidden_dim=config.hidden_dim,
18                 drop_lstm=config.dropout)
19         else:
20             self.encoder = LinearEncoder(label_size=config.label_size,
21                 input_dim=self.transformer.get_output_dim())
22         self.inferencer = LinearCRF(label_size=config.label_size, label2idx=
23             config.label2idx, add_iobes_constraint=config.add_iobes_constraint
24             ,
25                 idx2labels=config.idx2labels)
26         self.pad_idx = config.label2idx[PAD]

```

The embedding layer is useful for the model to understand the unique linguistic features in the Sinhala language. Applying contextual embedding within the model enables better named entity identifications, such as person, medical, creative work, organization, and location.

In our research, the pre-trained model **indicBERT** provides the contextual embeddings followed by a few steps. Initially it is necessary to load all the configuration files and pretrained transformer model. Then we need to adjust the position embeddings of a transformer model based on the maximum length of input tokens, which is denoted as `max_length`. Here, it will check whether the maximum length is more than the model’s mentioned value, then it will give the warning and make necessary adjustments. The model is loaded at runtime using Huggingface AutoModel, and it will output the contextual embeddings in the Sinhala language by referring to input sub-word tokens. A CRF layer is implemented as the inference layer in our transfer-learned model. This framework can be easily fine-tuned as necessary with our own dataset, and we have used the Sinhala dataset, which was extracted from Wikipedia. By default the hyperparameter tuning is not available, although the hyperparameters are configurable. This configuration allows us to use pre-trained static and contextual embedding as necessary. For example, **self.embedder_type** can be replaced with a pre-trained embedder, as shown below A.5.

For the training purposes, we have used datasets created from the explained processes in Section 4.1. These datasets are then loaded and tokenized to feed into the model. The tokenizing mechanism used for BERT models is known as **Wordpiece** tokenization. This algorithm decomposes words into sub-words. So unknown words can be represented from subword token IDs. The dataset consists of 3 separate text files, such as `train.txt`, `dev.txt`, and `test.txt` as shown in the below Appendix A.6. The annotated data is converted to **IOBES** format for better model learning (Ratinov and Roth 2009).

The main issue in using pre-training models is the truncation. Since neural networks need fixed-size input, truncation is inevitable. Hence there will be a potential loss in context. However, the context loss can be mitigated by using approaches like sliding windows, although we have not applied them here.

We can also customize the early stopping criteria as preferred so that models

are not trained unnecessarily. This will save computational resources during the training but also ensures that the NER model for Sinhala generalizes better for unseen Sinhala data. In our research we use precision as the early stopping criteria. So if the precision does not improve for **max_no_incre** consecutive epochs, then training is stopped to avoid overfitting and high resource usage. Early stopping increments can be changed via command-line arguments. During our research, we have applied early stopping criteria as below. An instance with random hyperparameters shown below.

```
!CUDA_LAUNCH_BLOCKING=1 python /content/drive/MyDrive/
    neural_crf/transformers_trainer.py \
--device=cuda:0 \
--dataset=MyData \
--model_folder=saved_models \
--embedder_type=ai4bharat/indic-bert \
--batch_size=4 \
--num_epochs=5 \
--learning_rate=3e-5 \
--fp16 1 \
--dropout 0.1 \
--max_no_incre 5 \
```

Before compiling a machine learning model, it is necessary to convert them into a numerical format from each instance, which contains words and their corresponding labels. Using the tokenizer as it is mentioned in the below code segment Appendix A.7, it will transform words into input IDs(subword tokens) and attention masks while mapping their original words with their corresponding IDs. In addition to that, it will do some processing to handle length mismatches while applying the padding mechanism. Finally, it will generate a list of dictionaries that contains input IDs, attention masks, and labels, which are useful for the model compilation.

4.3.1 Data Augmentation

In Sinhala NLP research, we face the issue of data scarcity. Data scarcity can potentially lead to class imbalance. Class imbalance is the unequal distribution of examples among predefined classes. Especially if large datasets are used in training, class imbalance can be a major issue, because then the model may become biased to predict the more frequent classes. This will give poor performance on low-frequency classes and result in low accuracy on NER tasks. NER datasets tend to contain a large amount of 'O' category tokens. Not only that, such unequal distribution can cause overfitting of the model.

Therefore, addressing class imbalance is necessary for the NER model accuracy to ensure that the model can effectively identify the relevant entity or tags in the text. It involves some techniques like data augmentation, changing the existing data to give more weight to the low-class distributions. Some of the augmentation techniques in NLP include synonym replacement, back translation, word order shuffling, and random insertion or deletion, which can be applied for NER tasks.

Among existing data augmentation techniques, back translation and synonym replacement are common in NLP tasks. However, synonym replacement does not work for the Sinhala language. Because, in Sinhala, synonyms do not always share the same meaning, where cultural context plays a significant role. So, if we replace the sentence with a synonym, it can lead to misclassification or confusion, which leads the model to struggle with named entity recognition. For instance, let's consider the proper noun in Sinhala called 'kołamba' (Colombo), which is considered the capital of Sri Lanka. If we try to replace synonyms and replace them with 'nagaraya' (city), it will change the context within the document.

Therefore, we have selected the back-translation technique to augment data in our research to mitigate the class imbalance issue. Back translation preserves semantics and label integrity of tokens, unlike random insertion and synonym replacement. A slight change in the meaning of the original token may generalize

the model.

For every back translation we need a pivot language. Bengali was selected as the pivot language. We chose Bengali as the middle (pivot) language for Sinhala back translation because both come from the Indo-Aryan language family, which means they share certain linguistic Features. This shared features helps to capture linguistic patterns in machine translation in an effective manner. For the dataset we chose the Bengali version of the **MultiCoNER/multiconer_v2** dataset (Fetahu, Z. Chen, et al. 2023), which has the exact set of labels as the Sinhala dataset, ensuring consistency between the datasets. The dataset was originally tagged with fine-grained entities, so the tags were again re-tagged with coarse grained categories.

We first filter out sentences with only the "**Other**" category to extract semantically rich examples for augmentation purposes. Then we further extract sentences from the selected token-label pairs to feed into the translation model. We have used **facebook/m2m100_418M** model (Fan et al. 2021) for the translation purposes. Translation involves the forward translation (**Sinhala** \rightarrow **Bengali**) and back translation (**Bengali** \rightarrow **Sinhala**) steps. Then the new tokens are combined with the corresponding label. Following code sample explains in the Appendix A.8 how we managed to apply back translation from Bengali to Sinhala.

However, back translation failed to translate all the words due to the complexity of the language. It is a common fact that back transliteration can introduce inconsistencies or errors due to linguistic complexities of language, such as spelling errors and pronunciations. Therefore, it is crucial to apply post-processing techniques to mitigate those inconsistencies in the augmented dataset. Through this, we can refine and enhance the model’s performance in the named entity recognition task. Once back translation was completed, we conducted the post-processing to refine the augmented dataset prior to the model training. Through that, we managed to translate all the words that were skipped by the

automatic translation process.

4.3.2 Hyperparameter Tuning

Hyperparameter tuning is an important part of any machine learning project. It involves finding the optimal set of parameters to achieve the best performance. Usually this tuning is carried out on a validation dataset. Then the selected parameters are used in the final model training. There are different techniques used for hyperparameter tuning, such as grid search and random search.

In this research, hyperparameter tuning is achieved through **Optuna** (Akiba et al. 2019) optimization. It uses Bayesian optimization to explore and determine the optimal hyperparameters for batch size, learning rate, and dropout rate. Here, it will discard poor trials early, allowing for faster convergence of the model configuration and it is beneficial for the NER task. Because of that, we have chosen Optuna among other techniques.

Unlike grid search, where the model runs through all hyper-parameters, Optuna uses previous model results to predict the best combination of hyper-parameters using Bayesian optimization. Hence Optuna is efficient compared to grid search. This method has shown significant results in NER compared to traditional techniques, and it will explain in the next chapter. We chose learning rate, batch size, and dropout as our hyperparameters for optimization. The optimization objective was defined as the average of validation and test F1 scores to balance overfitting and generalization. After 25 trials, the best hyper-parameter values are selected based on the objective mentioned above and how we managed to achieve that, as shown below. The objective remains the same for multilingual fine tuning and data augmentation experiments to maintain consistency of the research.

```

1 # Set hyperparameters in the config
2 args.learning_rate = trial.suggest_float("learning_rate", 1e-5, 1e-3, log=True
    )
3 args.batch_size = trial.suggest_categorical("batch_size", [8, 16, 32])
4 args.dropout = trial.suggest_float("dropout", 0.0, 0.5)
5 study = optuna.create_study(direction="maximize") # We want to maximize the
    combined score
6 study.optimize(objective, n_trials=25) # Run the optimization for 25 trials
7 # Save the results after the optimization
8 save_results_to_file(study, "trial_results.csv")
9 # Get the best trial and print the results
10 best_trial = study.best_trial

```

4.4 Implementation of the Multilingual Transfer Learning Model

The difference between multilingual and monolingual data is simply the datasets. In the multilingual setting, the same monolingual pre-trained model is trained with an extended dataset. The extension comes from the addition of Bengali data. We keep the testing data set untouched for comparison with monolingual training.

The Bengali dataset used for the multilingual training was taken from the **MultiCoNER/multiconer_v2** (Fetahu, Z. Chen, et al. 2023). The dataset is categorized into six different labels (Location (LOC) ,Creative Work (CW) ,Group (GRP) ,Person (PER) ,Product (PROD) ,Medical (MED)). These six categories are mapped to fine-grained categories for better NE recognition. These categories are consistent with existing monolingual data categories because during dataset extraction from Wikipedia, we have been using the same categories.

After loading each token (or word) from the dataset, it is analyzed to detect its named entity tag and categorized into relevant groups such as person, creative work, location, organization, etc., as shown below. The main category for the token or word is identified by referring to the mapping that we defined at the beginning. After assigning the detected new NER tags to the tokens (or words),

the updated multilingual dataset is combined with the original dataset to improve the performance in Sinhala NE tag identification.

```

1  # Function to map NER tags
2  def map_ner_tags(ner_tags):
3      mapped_tags = []
4      for tag in ner_tags:
5          if tag == "O":
6              mapped_tags.append(tag)
7          else:
8              prefix, entity = tag.split("-", 1) # Split "B-OtherPROD" -> "B",
              "OtherPROD"
9              mapped_tags.append(f"{prefix}-{mapping.get(entity, entity)}") #
              Map to coarse label
10     return mapped_tags

```

In the named entity recognition task, identifying the mapping from fine-grained to coarse-grained is crucial for the smooth entity classifications in the Sinhala language. Because fine-grained labels give details about subcategories specific to entities, and coarse-grained labels categorize them into larger groups. For example, if we consider visual work, musical work, artwork, and written work as fine-grained options in the coarse-grained creative work category, as it is shown below. Through this, we can easily adapt to the different contexts or applications when it comes to entity recognition. Not only that, it will help to improve the model's accuracy while considering a broader level than an abstract level.

```

1  from datasets import load_dataset
2  import json
3  import pandas as pd
4  ds = load_dataset("MultiCoNER/multiconer_v2", "Bangla_(BN)")
5  # Define mapping from fine-grained to coarse-grained labels
6  mapping = {
7      "Facility": "LOC", "OtherLOC": "LOC", "HumanSettlement": "LOC", "Station":
          "LOC",
8      "VisualWork": "CW", "MusicalWork": "CW", "WrittenWork": "CW", "ArtWork": "
          CW", "Software": "CW",
9      "MusicalGRP": "GRP", "PublicCORP": "GRP", "PrivateCORP": "GRP", "
          AerospaceManufacturer": "GRP",
10     "SportsGRP": "GRP", "CarManufacturer": "GRP", "ORG": "GRP",
11     "Scientist": "PER", "Artist": "PER", "Athlete": "PER", "Politician": "PER"
          , "Cleric": "PER",
12     "SportsManager": "PER", "OtherPER": "PER",
13     "Clothing": "PROD", "Vehicle": "PROD", "Food": "PROD", "Drink": "PROD", "
          OtherPROD": "PROD",
14     "Medication/Vaccine": "MED", "MedicalProcedure": "MED", "
          AnatomicalStructure": "MED",
15     "Symptom": "MED", "Disease": "MED"
16 }

```

Finally, we have applied hyperparameter tuning and data augmentation and evaluated the performances of each experiment using the matrices such as precision, accuracy, and F1 score. Those results were critically analyzed in the next chapter.

Chapter 5

Results and Analysis

In this chapter, the results that have been obtained during our research through several experiments on transfer learning for the NER task in Sinhala are described. The evaluation matrices we have selected to evaluate the performance in named entity recognition in Sinhala are explained under Section 5.1. Section 5.2 represents the results obtained from the monolingual transfer learning NER model for Sinhala, and Section 5.3 explains the multilingual transfer learning results in a descriptive manner.

5.1 Evaluation Metrics

Evaluation metrics are essential in checking the model performance quantitatively. Named entity recognition, by nature, is a classification task. Hence, we need classification-based evaluation metrics to indicate how well the model performs on classifying data into several classes, such as person, location, organization, etc. Now there are novel evaluation methods to evaluate the specific task of NER, but we avoid them because of their complexity of implementation. During our research, we have used 3 evaluation metrics, such as F1 score, precision, and recall.

In NER, evaluation can be considered as two ways: token-level predictions and entity-level predictions. During token-level predictions, it will focus on individual words or tokens within the text and check whether each token is correctly classified or not as it shown in the Figure 5.1 and Figure 5.2. But, in the entity-level predictions, evaluate the model's performance based on entire entities, and it is more complex than token-level prediction.

Sentence: මහින්ද රාජපක්ෂ කොළඹට ගියාය.
 Labels: B-PER I-PER B-LOC O

Figure 5.1: An example of token-level tagging scheme

Entities:

- මහින්ද රාජපක්ෂ → PER (Person)
- කොළඹට → LOC (Location)

Figure 5.2: An example of entity-level tagging schemes

5.1.1 Precision

The metric precision basically calculates the ratio of correctly predicted positive entities to the total predicted positive entities. Here, positivity refers to detecting an entity. A higher precision means the model's entity identification is accurate.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

False Positives (FP) - An incorrect prediction

True Positives (TP) - A correctly predicted entity

True Positives (TP) + False Positives (FP) - Total No of predicted entities

5.1.2 Recall

Recall is about how many of the actual entities in the data were correctly found by the model. A higher recall means the model is effective for identifying most of the entities in the Sinhala text, while a lower recall means that the model will miss some entity identifications.

$$\text{Recall} = \frac{\text{True Positives(TP)}}{\text{True Positives(TP)} + \text{False Negatives(FN)}}$$

5.1.3 F1- Score

The **F1-Score** is a metric which evaluates the balance between recall and precision, the harmonic mean value of the recall and precision. The harmonic value penalizes extreme values of recall and precision. This ensures that model doesn't trade off precision over recall or vice-versa. In simple terms. The F1 score indicates the performance of the model authentically.

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5.2 Experiments and Results on the Monolingual Model

Prior to the evaluation, it is necessary to train the model by using the extracted Sinhala named entity dataset, which was extracted from Wikipedia and covers six different categories, as explained in Chapter 4. As explained, we built the baseline Sinhala NER model following the proposed architecture, which was introduced by (Azeez and Surangika Ranathunga 2020b), which considers the CRF approach. Then, using the IndicBERT multilingual Indian language model as the source model, we applied transfer learning to predict the NE tags in Sinhala using our previously extracted Wiki dataset. After that, we performed hyperparameter tuning and data augmentation and evaluated each model's accuracy using the aforementioned evaluation metrics. For training, validation, and testing purposes, we used the 2668094, 1222781, and 1222781 numbers of tokens, respectively, covering 11 categories such as 'B-CW', 'B-GRP', 'B-LOC', 'B-PER', 'B-PROD', 'I-CW', 'I-GRP', 'I-LOC', 'I-PER', 'I-PROD', and 'Other'.

5.2.1 Results from the baseline model

The baseline model was trained on a Google Colab environment with an NVIDIA Tesla T4 high-RAM GPU. And the epoch numbers were changed, fixing the L1 regularization at 0.1, the L2 regularization at 1e-4 and enabling feature.possible transitions. After the first run the model was only learning to predict **Other** and

the macro-F1 was only 0.1.

At the early stage of training (epoch 15), the model gave low values for precision, recall, and F1-score, which were all at 0.10. It means the model is struggling to identify the entities with low epoch training. However, training reached to epoch 55, it showed a significant improvement on results, while precision increased up to 0.88, recall to 0.48, and the F1-score to 0.58. When it reached the 155th epoch, the model further enhanced its performance, achieving precision of 0.90, recall of 0.51, and an F1 score of 0.61. After the epoch was set to 155, the F1 score reached 0.61. So we can safely conclude that increasing the epoch number improves the baseline model’s performance, as shown in Table 5.1.

Metric	Epoch 15	Epoch 55	Epoch 155
Precision (Macro Avg)	0.10	0.88	0.90
Recall (Macro Avg)	0.10	0.48	0.51
F1-score (Macro Avg)	0.10	0.58	0.61

Table 5.1: Comparison of Macro-Averaged Precision, Recall, and F1-Score across three evaluation instances

The evaluations were based on tokens, as shown in Figure 5.3. The macro average values consider each class equally by calculating the metrics for each class and then averaging those values. This figure shows the test performances on the baseline CRF model for different categories, such as person, location, organization, creative work, etc. Among them, the "Other" category shows the remarkable values for precision, recall, and F1-score, 0.99, 1.00, and 1.00, respectively, with the support of a large sample of 93,161. The "I-CW" category shows the lowest recall value at 0.10. Depending on the results, it is clear that the model can accurately detect some categories, but for others we still need optimizations. We used these results as our baseline model to compare the performance of the transfer-learned model in experiments in both monolingual and multilingual approaches.

Final Test Performance:				
	precision	recall	f1-score	support
B-CW	0.92	0.38	0.54	210
B-GRP	0.96	0.77	0.86	119
B-LOC	0.98	0.87	0.92	339
B-PER	0.79	0.39	0.52	270
B-PROD	1.00	0.50	0.67	2
I-CW	0.75	0.10	0.18	87
I-GRP	1.00	0.18	0.30	17
I-LOC	0.96	0.55	0.70	93
I-PER	0.67	0.34	0.45	173
Other	0.99	1.00	1.00	93161
accuracy			0.99	94471
macro avg	0.90	0.51	0.61	94471
weighted avg	0.99	0.99	0.99	94471

Figure 5.3: Baseline CRF model performance

5.2.2 Results of the monolingual transfer learning model

From 2018 onwards, there has been a significant impact on the model’s performance when the transfer learning technique is used in named entity recognition. Sinhala is considered a low-resource, morphologically rich language, and due to that reason, most deep learning models have failed due to data scarcity. Transfer learning allows us to transfer knowledge from the pre-trained models on larger datasets to predict NE tags in low-resource languages like Sinhala. Additionally, it helps in capturing linguistic nuances and contextual information, which are crucial for detecting named entities in Sinhala. Therefore, it is necessary to find the most suitable pre-trained model, and we have chosen IndicBERT as our source model.

We have used the Wikipedia-extracted dataset of named entities in Sinhala covering six categories, as explained in the previous chapter, for the target model’s training dataset. First we compiled the IndicBERT model and then trained it with the aforementioned dataset using the learning rate of 3×10^{-5} , batch size of 4, and dropout rate of 0.1. Finally, observe the performances using precision, recall, and F1 score. The evaluation metrics were recorded on the Dev sets and Test sets respectively.

Here, we have observed that it has very low results because there were instances where early dropouts occurred due to poor learning. To overcome that, we have used hyperparameter tuning for our transfer-learned model as our 2nd experiment. Because our pre-trained model parameters were designed to predict

Indian language NE, they may not be effective for Sinhala. Therefore, we have decided to go with hyperparameter tuning with our model.

During our research, we have experimented with three different hyperparameter tuning approaches, such as Optuna hyperparameter tuning, grid search, and random search. Among them, we have observed that the random search gives optimal hyperparameters for our monolingual transfer learning. Here, the learning rate, batch size, and dropout rate were chosen as hyperparameters. While running around 10 iterations for 15 hours, the model gave those hyperparameter values as a learning rate of $1e-5$, a batch size of 16, and a dropout rate of 0.1. After applying hyperparameter tuning, the best performances on the test sets were obtained as shown in Table 5.2.

Precision	Recall	F1
47.49	0.24	0.47

Table 5.2: Best test set performance based on precision in a monolingual transfer learning .

With the model’s obtained precision value of 47.49, it indicates that nearly 50% of the entities in Sinhala identified by the model are correct, which is a fairly good achievement for our transfer learning approach. During our research, we are trying to figure out the applicability of transfer learning on Sinhala, and we can mention that applying hyperparameter tuning is a good start for that approach. We received 0.47 for the F1 score, which indicates that the model is effective to some extent, yet it struggles to understand some of the entities in the text.

The main reason for this weak performance can be the class imbalance. An overwhelming majority of the dataset tokens are tagged with the **Other** token, as shown in Table 4.1. As a remedy, we applied data augmentation to selected examples from training data, and it was considered our 3rd experiment. The remedy slightly increased the precision, as it shown in Table 5.3.

These results indicate that when we applied the augmentation, the precision improved slightly to 48.51, with a slight increase of 1.02, but the recall and F1 score decreased compared to the values of the monolingual transfer learning

model without any augmentation. It means while applying augmentation, it helped to identify some entities more accurately than other experiments. When we execute 8 number of epochs, it indicates that the best F1 score for the model with augmentation is recorded as 1.42, and without augmentation it is recorded as 1.21. So we can conclude that after the augmentation, the model can make accurate predictions and detect entities slightly better.

Instance	Precision	Recall	F1
Sinhala + Augmented Data	48.51	0.14	0.28

Table 5.3: Best test set performance based on precision for transfer learning in a monolingual model with augmentation.

5.3 Experiments and Results on the Multilingual Model

In this section, we mention the experiments which we conducted during the research to evaluate the performance of our multilingual model, which built upon IndicBERT. Here we used the dataset that covered both Sinhala and Bengali language named entity tags, and it contains 5,113,656 tokens and 393,509 tokens, respectively. By critically analyzing the measurements in both the hyperparameter-tuned and augmented models, we aim to demonstrate the multilingual model’s effectiveness in Sinhala NE predictions.

The multilingual setting differs from the monolingual setting only in the dataset. Here the original Sinhala dataset, which we extracted through Wikipedia, is combined with the Bengali dataset (Fetahu, Z. Chen, et al. 2023).

The best precision was recorded with a learning rate of 4.663×10^{-5} , a batch size of 32, and a dropout rate of 0.148 after experimenting with various hyperparameters (see Table 5.4). The precision of 41.58 is still lower than the previous tables 5.2 and 5.3 precision values. It suggests that focusing on Sinhala data allowed the model to better understand the Sinhala nuances of the language, which helps to predict named entities accurately, and it will improve further with proper augmentation. But the difference can be due to the different language

examples in the training data. The provided multilingual dataset consisted of 7 labels in large density. Ideally the model would learn quickly with the added examples, but this effect may trade off with the difficulties related to language diversity. When we executed 10 numbers of epochs, it recorded the best F1 score as 1.39 when we applied hyperparameter tuning for the multilingual transfer learning model.

Precision	Recall	F1
41.58	0.11	0.237

Table 5.4: Best test set performance based on precision for transfer learning in a multilingual model.

The precision alone isn't enough to evaluate a model. Recorded F1 scores were exceedingly low, which signifies the low recall values. So the model is very poor at detecting entities. Earlier in the development phase, input sequences were truncated to feed into the model. This may have incur a significant context loss. Context is importance for the NER task. So this context loss can be a contributing factor to the low F1 score.

Chapter 6

Conclusion

6.1 Summary

The objective of this research is to investigate the applicability of transfer learning for Sinhala NER. Current Sinhala NER work involves a CRF approach to recognize named entities. However, CRF performance relies on involves using labeled data. Manual annotation of labeled data is time-consuming for Sinhala due to its complex syntax and morphological structure. There are few freely available named entity datasets for Sinhala, but all of them cover three categories. During our research, we want to find out performances on much larger named entity categories.

Therefore, we have to extract the Sinhala named entity list from scratch because we cannot use existing ones, as they focused on three groups, which can lead to misclassifications of entities. We have used an automated weakly supervised approach to extract Sinhala named entity data from Wikipedia articles, including six categories such as person, location, organization, creative work, group, and medical. We have used that the extracted dataset for training purposes on transfer learning approaches. During our research, we have checked the applicability of transfer learning on monolingual and multilingual approaches. Consequently, multilingual model training was conducted with the Bengali dataset.

In transfer learning, performance depends on the source model selection, and for our research, we have used IndicBERT as our source model due to its similarities with the Sinhala language. It was trained with a large Indian language dataset for the named entity recognition task. Through source model knowledge,

it will predict target Sinhala entities, and it helps us to tackle the low-resource problem. Then we trained both the monolingual and multilingual approaches with hyperparameter tuning and data augmentation under several experiments. Finally, they were evaluated with the test dataset, and precision, recall, and F1 score were recorded to compare it with the baseline NER model to evaluate the effectiveness of the transfer learning approach.

The monolingual transfer learning model, which was evaluated on an augmented dataset, showed 48.51% precision. The multilingual model showed 41.58% precision, which is slightly less than the monolingual approach. In machine learning models, we have a fixed length, and if a given sentence is much larger than that, then we have to apply truncating to handle that. And the above-mentioned precision was recorded after applying truncation of the input sentence. The precision and low F1 score could have improved if the context loss was avoided through a mechanism like sliding windows, which we used in truncation.

The multilingual model could have shown a better performance if trained for a high number of epochs with a much higher GPU environment. Perhaps then the model would learn patterns involving the Bengali language. In fact, in hyperparameter tuning, the number of epochs used in all experiments was fixed at 15 due to time constraints. Hence, using a much larger number of epochs can improve the low F1 scores and precision.

We provide an answer for an important RQ raised in Chapter 1.

- The most effective hyper-parameters for our extracted Sinhala named entity dataset are explained in the Implementation chapter with justification for each selection.
- How does the application of different data augmentation techniques improve the overall performance of the entity recognition in the Sinhala NER?

In the monolingual setting, we observed that data augmentation improved the precision by 2.15% and F1-Score by 17.36%. And with better hyperparameter tuning (like increasing the number of epochs from the fixed 15)

there is a lot of room for improvement.

6.2 Limitations

The dataset extracted for this research used Wikipedia as the source. Wikipedia content has noise and few context compared to natural context like newspaper articles. Due to that, we were unable to capture more named entities for most of the newly introduced categories. We tried to use manual annotation in this research to add context to our dataset. But manual annotation is a time-consuming process. Apart from annotation, manual annotation involves human supervision (using measures like Kappa(J. Cohen 1960)) as well hence, we didn't proceed with the manual annotation.

6.3 Future directions

In this research we have employed a BERT encoder and a CRF layer to build the model. But there is flexibility in the model to employ a BiLSTM layer instead of a linear encoder. This is an interesting experiment. The dataset used in this research was created by using the method in (Krishnan et al. 2021). The method involves using Wikipedia as a source to extract data. The (ibid.) work suggests using some language-specific rules as enhancements in the dataset. We did not apply those rules in this research. Hence the dataset can be improved, and results can be improved with the assistance from the linguistic specialist.

Another area for improvement is the Wikipedia-extracted dataset. The dataset used in this research has a class imbalance due to poor entity distribution in most of the Sinhala Wikipedia articles. There are an overwhelming number of tokens tagged with the **Other** tag. An effort can be made to mitigate class imbalance by using a dataset with a wider context, like (Common Crawl n.d.).

We are only using Bengali as a supporting language for multilingual fine-tuning. Hence another exciting experiment will be to try out different language combinations in multilingual fine-tuning to select the best-performing combina-

tion of languages in transfer learning for Sinhala NER.

Bibliography

- Akiba, Takuya et al. (2019). *Optuna: A Next-generation Hyperparameter Optimization Framework*. arXiv: 1907.10902 [cs.LG]. URL: <https://arxiv.org/abs/1907.10902>.
- Azeez, Rameela and Surangika Ranathunga (2020a). “Fine-Grained Named Entity Recognition for Sinhala”. In: *2020 Moratuwa Engineering Research Conference (MERCon)*, pp. 295–300. DOI: 10.1109/MERCon50084.2020.9185296.
- (2020b). “Fine-Grained Named Entity Recognition for Sinhala”. In: *2020 Moratuwa Engineering Research Conference (MERCon)*, pp. 295–300. DOI: 10.1109/MERCon50084.2020.9185296.
- Brants, Thorsten (Apr. 2000). “TnT – A Statistical Part-of-Speech Tagger”. In: *Sixth Applied Natural Language Processing Conference*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 224–231. DOI: 10.3115/974147.974178. URL: <https://aclanthology.org/A00-1031>.
- Chen, Shiqi et al. (2021). “Low-Resource Named Entity Recognition via the Pre-Training Model”. In: *Symmetry* 13.5, p. 786. DOI: 10.3390/sym13050786. URL: <https://doi.org/10.3390/sym13050786>.
- Cohen, Jacob (1960). “A Coefficient of Agreement for Nominal Scales”. In: *Educational and Psychological Measurement* 20.1, pp. 37–46. DOI: 10.1177/001316446002000104.
- Common Crawl (n.d.). *Common Crawl*. Accessed: 2025-04-26. URL: <https://commoncrawl.org/>.
- Conneau, Alexis et al. (Jan. 2020). “Unsupervised Cross-lingual Representation Learning at Scale”. In: pp. 8440–8451. DOI: 10.18653/v1/2020.acl-main.747.

- Cotterell, Ryan and Kevin Duh (Nov. 2017). “Low-Resource Named Entity Recognition with Cross-lingual, Character-Level Neural Conditional Random Fields”. In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Ed. by Greg Kondrak and Taro Watanabe. Taipei, Taiwan: Asian Federation of Natural Language Processing, pp. 91–96. URL: <https://aclanthology.org/I17-2016>.
- Dahanayaka, J. K. and A. R. Weerasinghe (2014). “Named entity recognition for Sinhala language”. In: *2014 14th International Conference on Advances in ICT for Emerging Regions (ICTer)*, pp. 215–220. DOI: 10.1109/ICTER.2014.7083904.
- Dahanayaka, Jinadi and Ruwan Weerasinghe (Dec. 2014). “Named Entity Recognition for Sinhala Language”. In: DOI: 10.1109/ICTER.2014.7083904.
- Devlin, Jacob et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs.CL]. URL: <https://arxiv.org/abs/1810.04805>.
- Dhamecha, Tejas et al. (Nov. 2021). “Role of Language Relatedness in Multilingual Fine-tuning of Language Models: A Case Study in Indo-Aryan Languages”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Ed. by Marie-Francine Moens et al. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, pp. 8584–8595. DOI: 10.18653/v1/2021.emnlp-main.675. URL: <https://aclanthology.org/2021.emnlp-main.675>.
- Doddapaneni, Sumanth et al. (2023). *Towards Leaving No Indic Language Behind: Building Monolingual Corpora, Benchmark and Models for Indic Languages*. arXiv: 2212.05409 [cs.CL]. URL: <https://arxiv.org/abs/2212.05409>.
- Fan, Angela et al. (2021). “Beyond English-Centric Multilingual Machine Translation”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

- Fernando, S. and S. Ranathunga (May 2018). “Evaluation of different classifiers for Sinhala POS tagging”. In: *2018 Moratuwa Engineering Research Conference (MERCon)*. IEEE, pp. 96–101.
- Fernando, S., S. Ranathunga, et al. (Dec. 2016). “Comprehensive part-of-speech tag set and SVM based POS tagger for Sinhala”. In: *Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WS-SANLP2016)*, pp. 173–182.
- Fetahu, Besnik, Zhiyu Chen, et al. (2023). “MultiCoNER v2: a Large Multilingual dataset for Fine-grained and Noisy Named Entity Recognition”. In.
- Fetahu, Besnik, Sudipta Kar, et al. (2023). “SemEval-2023 Task 2: Fine-grained Multilingual Named Entity Recognition (MultiCoNER 2)”. In: *Proceedings of the 17th International Workshop on Semantic Evaluation (SemEval-2023)*. Association for Computational Linguistics.
- Jayasinghe, K.L. (2017). *Bootstrapping Sinhala Named Entities for NLP Applications*. Unpublished paper. URL: <https://dl.ucsc.cmb.ac.lk/jspui/bitstream/123456789/4195/1/2014CS051.pdf>.
- Jie, Allan (2020). *PyTorch-Neural-CRF: Flexible BERT-BiLSTM-CRF Framework*. https://github.com/allanj/pytorch_neural_crf. Accessed: 2023-11-15.
- Kakwani, Divyanshu et al. (2020). “IndicNLP Suite: Monolingual Corpora, Evaluation Benchmarks and Pre-trained Multilingual Language Models for Indian Languages”. In: *Findings of EMNLP*.
- Khanuja, Simran et al. (2021). “MuRIL: Multilingual Representations for Indian Languages”. In: *ArXiv abs/2103.10730*. URL: <https://api.semanticscholar.org/CorpusID:232290691>.
- Krishnan, Aravind et al. (Sept. 2021). “Employing Wikipedia as a resource for Named Entity Recognition in Morphologically complex under-resourced languages”. In: *Proceedings of the 14th Workshop on Building and Using Comparable Corpora (BUCC 2021)*. Ed. by Reinhard Rapp, Serge Sharoff, and

- Pierre Zweigenbaum. Online (Virtual Mode): INCOMA Ltd., pp. 28–39. URL: <https://aclanthology.org/2021.bucc-1.5/>.
- Lafferty, John D., Andrew McCallum, and Fernando C. N. Pereira (2001). “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 282–289. ISBN: 1558607781.
- Lample, Guillaume et al. (June 2016). “Neural Architectures for Named Entity Recognition”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Kevin Knight, Ani Nenkova, and Owen Rambow. San Diego, California: Association for Computational Linguistics, pp. 260–270. DOI: 10.18653/v1/N16-1030. URL: <https://aclanthology.org/N16-1030/>.
- Manamini, S.A.P.M. et al. (Apr. 2016). “Ananya - a Named-Entity-Recognition (NER) system for Sinhala language”. In: pp. 30–35. DOI: 10.1109/MERCon.2016.7480111.
- Murthy, Rudra and Pushpak Bhattacharyya (Jan. 2018). “A Deep Learning Solution to Named Entity Recognition”. In: pp. 427–438. ISBN: 978-3-319-75476-5. DOI: 10.1007/978-3-319-75477-2_30.
- Ostendorff, M. and G. Rehm (2023). “Efficient Language Model Training through Cross-Lingual and Progressive Transfer Learning”. In: *arXiv.org*. DOI: 10.48550/arXiv.2301.09626.
- Pan, Sinno Jialin and Qiang Yang (2010). “A Survey on Transfer Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 22.10, pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.
- Patil, Nita, Ajay Patil, and Bhausahab Pawar (Jan. 2020). “Named Entity Recognition using Conditional Random Fields”. In: *Procedia Computer Science* 167, pp. 1181–1188. DOI: 10.1016/j.procs.2020.03.431.

- Al-Qurishi, Muhammad Saleh and Riad Souissi (Dec. 2021). “Arabic Named Entity Recognition Using Transformer-based-CRF Model”. In: *Proceedings of the 4th International Conference on Natural Language and Speech Processing (ICNLSP 2021)*. Ed. by Mourad Abbas and Abed Alhakim Freihat. Trento, Italy: Association for Computational Linguistics, pp. 262–271. URL: <https://aclanthology.org/2021.icnls-1.31/>.
- Ratinov, Lev and Dan Roth (June 2009). “Design Challenges and Misconceptions in Named Entity Recognition”. In: *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*. Ed. by Suzanne Stevenson and Xavier Carreras. Boulder, Colorado: Association for Computational Linguistics, pp. 147–155. URL: <https://aclanthology.org/W09-1119/>.
- Rijhwani, Shruti et al. (July 2020). “Soft Gazetteers for Low-Resource Named Entity Recognition”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Ed. by Dan Jurafsky et al. Online: Association for Computational Linguistics, pp. 8118–8123. DOI: 10.18653/v1/2020.acl-main.722. URL: <https://aclanthology.org/2020.acl-main.722>.
- Sabane, Maithili et al. (May 2023). “Enhancing Low Resource NER using Assisting Language and Transfer Learning”. In: pp. 1666–1671. DOI: 10.1109/ICAAIC56838.2023.10141204.
- Saha, Sriparna and Asif Ekbal (May 2013). “Combining multiple classifiers using vote based classifier ensemble technique for named entity recognition”. In: *Data Knowledge Engineering* 85, pp. 15–39. DOI: 10.1016/j.datak.2012.06.003.
- Senevirathne, K.U. et al. (2015). “Conditional Random Fields based Named Entity Recognition for Sinhala”. In: *2015 IEEE 10th International Conference on Industrial and Information Systems (ICIIS)*, pp. 302–307. DOI: 10.1109/ICIINFS.2015.7399028.

- Singh, Thoudam Doren, Nameirakpam Basanta, and Potsangbam Kumar Singh (2020). “Named Entity Recognition for Manipuri Using Support Vector Machine”. In: *Journal of Intelligent Systems* 29.1, pp. 123–135. DOI: 10.1515/jisys-2019-0123. URL: <https://doi.org/10.1515/jisys-2019-0123>.
- Subbarao, K. (Jan. 2012). *South Asian Languages: A Syntactic Typology, supplementary web material to the publication South Asian Languages: A Syntactic Typology, Cambridge University Press, New York: New Delhi, 2012*.
- Tan, Chuanqi et al. (Oct. 2018). “A Survey on Deep Transfer Learning”. In: *Artificial Neural Networks and Machine Learning – ICANN 2018: 27th International Conference on Artificial Neural Networks, Proceedings, Part III*. Vol. 11141. Lecture Notes in Computer Science. Rhodes, Greece: Springer, pp. 270–279. DOI: 10.1007/978-3-030-01424-7_27. URL: https://link.springer.com/chapter/10.1007/978-3-030-01424-7_27.
- Torge, Sunna et al. (May 2023). “Named Entity Recognition for Low-Resource Languages - Profiting from Language Families”. In: *Proceedings of the 9th Workshop on Slavic Natural Language Processing 2023 (SlavicNLP 2023)*. Ed. by Jakub Piskorski et al. Dubrovnik, Croatia: Association for Computational Linguistics, pp. 1–10. DOI: 10.18653/v1/2023.bsnlp-1.1. URL: <https://aclanthology.org/2023.bsnlp-1.1>.
- Vaswani, Ashish et al. (2017). “Attention is All you Need”. In: *Neural Information Processing Systems*. URL: <https://api.semanticscholar.org/CorpusID:13756489>.
- Wijesinghe, W.M.S.K. and Muditha Tissera (2022). “Sinhala Named Entity Recognition Model: Domain-Specific Classes in Sports”. In: *2022 4th International Conference on Advancements in Computing (ICAC)*, pp. 138–143. DOI: 10.1109/ICAC57685.2022.10025148.
- Wu, Minghao, Fei Liu, and Trevor Cohn (Oct. 2018). “Evaluating the Utility of Hand-crafted Features in Sequence Labelling”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Ed. by

- Ellen Riloff et al. Brussels, Belgium: Association for Computational Linguistics, pp. 2850–2856. DOI: 10.18653/v1/D18-1310. URL: <https://aclanthology.org/D18-1310>.
- Yang, Zhilin, Ruslan Salakhutdinov, and William W. Cohen (2017). “Transfer Learning for Sequence Tagging with Hierarchical Recurrent Networks”. In: *CoRR* abs/1703.06345. arXiv: 1703.06345. URL: <http://arxiv.org/abs/1703.06345>.
- Zhou, GuoDong and Jian Su (July 2002). “Named Entity Recognition using an HMM-based Chunk Tagger”. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Ed. by Pierre Isabelle, Eugene Charniak, and Dekang Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, pp. 473–480. DOI: 10.3115/1073083.1073163. URL: <https://aclanthology.org/P02-1060/>.

APPENDICES

A Appendix A Implementation

Listing A.1: Map GKG Type to Entity Function

```
# Function: Map GKG Type to Entity
def map_gkg_type_to_entity(gkg_types):
    GKG_TO_ENTITY_MAP = {
        "Place": "LOC",
        "AdministrativeArea": "LOC",
        "City": "LOC",
        "Country": "LOC",
        "Organization": "GRP",
        "Corporation": "GRP",
        "EducationalOrganization": "GRP",
        "Person": "PER",
        "CreativeWork": "CW",
        "Book": "CW",
        "Movie": "CW",
        "TVSeries": "CW",
        "Product": "PROD",
        "MedicalEntity": "MED",
    }

    for gkg_type in gkg_types:
        if gkg_type in GKG_TO_ENTITY_MAP:
            return GKG_TO_ENTITY_MAP[gkg_type]
```

```

    return "Other"

def get_entity_type_from_kg(english_title, api_key):
    url = "https://kgsearch.googleapis.com/v1/entities:
        search"

    params = {
        "query": english_title,
        "key": api_key,
        "limit": 1,
        "languages": "en"
    }

    response = requests.get(url, params=params)
    data = response.json()

    entity_types = []
    if "itemListElement" in data:
        for result in data["itemListElement"]:
            entity_type = result.get("result", {}).get("@type", [])
            entity_types.extend(entity_type)

    return map_gkg_type_to_entity(entity_types)

```

Listing A.2: Fetching wikipedia content

```

# Function to fetch and annotate Wikipedia content
def fetch_wikipedia_content_sinhala(title, result):
    url = f"https://si.wikipedia.org/wiki/{title}"
    try:
        response = requests.get(url, headers={'User-Agent':
            'YourAppName/1.0 (your_email@example.com)'})
        if response.status_code == 200:

```

```

soup = BeautifulSoup(response.text, 'html.
    parser')
content = soup.find('div', {'class': 'mw-parser
    -output'})
paragraphs = content.find_all('p')
annotated_content = []
for p in paragraphs:
    text = p.get_text() # Extract text content
    sentences = nltk.sent_tokenize(text) #
        Sentence tokenization
    for sentence in sentences:
        # Tokenize words in the sentence
        tokens = list(
            tokenize_with_multiword_entities(
                sentence, result)) # Tokenize
                properly
        # Add word tokens and their labels
        for word, label in tokens:
            annotated_content.append(f"{word}\t
                {label}")
        # Check if the sentence ends with
            punctuation and insert a blank line
        if sentence[-1] in [',', '?', '!']:
            annotated_content.append("") #
                Insert a blank line
    return "\n".join(annotated_content)
else:
    return f"Failed to retrieve the article! Status
        code: {response.status_code}"
except Exception as e:
    return f"An error occurred: {str(e)}"

```


Listing A.3: Feature extraction from words

```

def word2features(sent, i):
    word = sent[i][0]
    postag = sent[i][1]
    features = [
        'bias',
        'word.lower=' + word.lower(),
        # 'word[-3:]=' + word[-3:],
        'word[-2:]=' + word[-2:],
        'word.isupper=%s' % word.isupper(),
        'word.istitle=%s' % word.istitle(),
        'word.isdigit=%s' % word.isdigit(),
        'word.position=%d' % i,
        'postag=' + postag,
        'word.in_gazetteer=%s' % (word in gazetteer)
    ]
    for category, entities in gazetteer.items():
        if word.lower() in entities:
            features.append(f'word.in_{category}={True}')
        else:
            features.append(f'word.in_{category}={False}')
    if i > 0:
        word1 = sent[i-1][0]
        postag1 = sent[i-1][1]
        features.extend([
            '-1:word.lower=' + word1.lower(),
            '-1:word.istitle=%s' % word1.istitle(),
            '-1:word.isupper=%s' % word1.isupper(),
            '-1:postag=' + postag1,
        ])
    else:

```

```

        features.append('BOS')
    if i < len(sent)-1:
        word1 = sent[i+1][0]
        postag1 = sent[i+1][1]
        features.extend([
            '+1:word.lower=' + word1.lower(),
            '+1:word.istitle=%s' % word1.istitle(),
            '+1:word.isupper=%s' % word1.isupper(),
            '+1:postag=' + postag1,
        ])
    else:
        features.append('EOS')

    # Add clue word features
    prev_word = sent[i-1][0] if i > 0 else ''
    next_word = sent[i+1][0] if i < len(sent) - 1 else ''
    features.extend(clue_word_feature(prev_word, next_word)
        )

    return features

```

Listing A.4: Hyperparameter Tuning for the baseline CRF

```

param_grid = {
    'c1': [0.1, 0.5, 1.0, 1.5], # L1 regularization
    'strengths',
    'c2': [1e-4, 1e-3, 1e-2], # L2 regularization
    'strengths',
    'max_iterations': [100, 200, 300],
    'feature.possible_transitions': [True, False]
}

best_score = 0
best_params = None

```

```

# Generate all parameter combinations
param_combinations = itertools.product(
    param_grid['c1'],
    param_grid['c2'],
    param_grid['max_iterations'],
    param_grid['feature.possible_transitions']
)
for c1, c2, max_iter, transitions in param_combinations:
    print(f"\nTesting params: c1={c1}, c2={c2}, max_iter={
        max_iter}, transitions={transitions}")
    # Initialize and train model
    trainer = pycrfsuite.Trainer(verbose=False)
    for xseq, yseq in zip(X_train, y_train):
        trainer.append(xseq, yseq)
    trainer.set_params({
        'c1': c1,
        'c2': c2,
        'max_iterations': max_iter,
        'feature.possible_transitions': transitions
    })
    # Train (use temporary file)
    temp_model = "temp.crfsuite"
    trainer.train(temp_model)
    # Validate
    tagger = pycrfsuite.Tagger()
    tagger.open(temp_model)
    y_val_pred = [tagger.tag(xseq) for xseq in X_val]
    # Calculate F1 score (micro-averaged)
    current_score = f1_score(
        [label for sent in y_val for label in sent],
        [label for sent in y_val_pred for label in sent],

```

```

        average='micro'
    )

```

Listing A.5: Trasnformer Architecture

```

class TransformersEmbedder(nn.Module):
    def __init__(self, transformer_model_name: str,
                  max_length: int = 512):
        super().__init__()
        logger.info(f"[Model_Info] Loading pretrained
                     language_model_{transformer_model_name}")
        # 1. Load config
        config = AutoConfig.from_pretrained(
            transformer_model_name)
        config.output_hidden_states = False
        config.return_dict = True
        # 2. Load model
        self.model = AutoModel.from_pretrained(
            transformer_model_name, config=config)
        # 3. Resize position embeddings if needed
        model_max_length = getattr(config, "
                                     max_position_embeddings", 512)
        if max_length > model_max_length:
            logger.warning(f"Model only supports {
                           model_max_length} tokens, but requested {
                           max_length}")
            logger.info(f"Extending position embeddings from
                         {model_max_length} to {max_length}")
            self.model.resize_position_embeddings(max_length)
        self.device = torch.device('cuda' if torch.cuda.
                                     is_available() else 'cpu')
        self.to(self.device)

```

```

def get_output_dim(self):
    return self.model.config.hidden_size

def forward(self, subword_input_ids: torch.Tensor,
             orig_to_token_index: torch.LongTensor, ##
             batch_size * max_seq_leng
             attention_mask: torch.LongTensor) -> torch.
    Tensor:
    """
    :param subword_input_ids: (batch_size x
        max_wordpiece_len) input token IDs
    :param orig_to_token_index: (batch_size x
        max_sent_len) indices mapping each original word
        to a subword token
    :param attention_mask: (batch_size x
        max_wordpiece_len)
    :return: word-level representations (batch_size x
        max_sent_len x hidden_size)
    """
    subword_rep = self.model(**{"input_ids":
        subword_input_ids, "attention_mask":
        attention_mask}).last_hidden_state
    batch_size, _, rep_size = subword_rep.size()
    _, max_sent_len = orig_to_token_index.size()
    # select the word index.
    word_rep = torch.gather(subword_rep[:, :, :], 1,
        orig_to_token_index.unsqueeze(-1).expand(
        batch_size, max_sent_len, rep_size))
    return word_rep

```

Listing A.6: Configurations of the Neural Network

```
class Config:
```

```

def __init__(self, args) -> None:
    """
    Construct the arguments and some hyperparameters
    :param args:
    """
    # Model hyper parameters
    self.embedder_type = args.embedder_type if "
        embedder_type" in args.__dict__ else None
    self.add_iobes_constraint = args.
        add_iobes_constraint
    # Data specification
    self.dataset = args.dataset
    self.train_file = "/content/drive/MyDrive/
        neural_crf/data/" + self.dataset + "/train.txt"
    self.dev_file = "/content/drive/MyDrive/neural_crf/
        data/" + self.dataset + "/dev.txt"
    self.test_file = "/content/drive/MyDrive/neural_crf
        /data/" + self.dataset + "/test.txt"
    self.train_num = args.train_num
    self.dev_num = args.dev_num
    self.test_num = args.test_num
    # Training hyperparameter
    self.model_folder = args.model_folder
    self.optimizer = args.optimizer.lower()
    self.learning_rate = args.learning_rate
    self.momentum = args.momentum if "momentum" in args
        __dict__ else None
    self.l2 = args.l2
    self.num_epochs = args.num_epochs
    self.use_dev = True
    self.batch_size = args.batch_size

```

```

self.clip = 5
self.lr_decay = args.lr_decay
self.device = torch.device(args.device) if "device"
    in args.__dict__ else None
self.max_no_incre = args.max_no_incre
self.max_grad_norm = args.max_grad_norm if "
    max_grad_norm" in args.__dict__ else None
self.fp16 = args.fp16 if "fp16" in args.__dict__
    else None

```

Listing A.7: Conversion of instances to features

```

def convert_instances_to_features(instances: List[
    Instance],

                                tokenizer:
                                    PreTrainedTokenizerFast
                                ,
                                label2idx: Dict[
                                    str, int]) ->
                                    List[Dict]:

    features = []
    skipped = 0
    for inst in instances:
        words = inst.ori_words
        labels = inst.labels
        if len(words) != len(labels):
            raise ValueError(f"Word/label_length_mismatch: {
                len(words)} words vs {len(labels)} labels")
        # Map labels to IDs
        label_ids = [label2idx.get(label, -100) for label
            in labels]
        encoding = tokenizer(

```

```

        words,
        is_split_into_words=True,
        truncation=True,
        padding=True, # padding handled later in
                       collate_fn
        return_offsets_mapping=True,
        max_length=512
    )

    input_ids = encoding["input_ids"]
    attention_mask = encoding["attention_mask"]
    offset_mapping = encoding["offset_mapping"]
    # Map word indices to subword token indices
    word_ids = encoding.word_ids()
    orig_to_tok_index = []
    orig_to_label = [] # This will store the labels
                       for the original words
    last_word_idx = None
    for i, word_idx in enumerate(word_ids):
        if word_idx is not None and word_idx !=
            last_word_idx:
            orig_to_tok_index.append(i)
            orig_to_label.append(label_ids[word_idx])
            last_word_idx = word_idx
    # Truncation might cut off some words
    if len(orig_to_tok_index) != len(words):
        skipped += 1
        continue
    features.append({
        "input_ids": input_ids,
        "attention_mask": attention_mask,
        "orig_to_tok_index": orig_to_tok_index,

```



```

        "label_ids": orig_to_label,
        "word_seq_len": len(orig_to_tok_index),
    })
    return features

```

Listing A.8: Back Translation in Data Augmentation

```

def back_translate_dataset(sentences, translator,
    max_workers=4):
    """Back-translate sentences using thread pool for speed
    ."""
    def process(sentence):
        return translator.back_translate(sentence)[0]
    with ThreadPoolExecutor(max_workers=max_workers) as
        executor:
        return list(executor.map(process, sentences))
class M2MBackTranslator:
    def __init__(self, src='si', pivot='bn', device=None):
        self.src = src
        self.pivot = pivot
        self.device = device or ('cuda' if torch.cuda.
            is_available() else 'cpu')

        self.model_name = "facebook/m2m100_418M"
        self.tokenizer = M2M100Tokenizer.from_pretrained(
            self.model_name)
        self.model = M2M100ForConditionalGeneration.
            from_pretrained(self.model_name).to(self.device)
    def translate(self, texts, src_lang, tgt_lang):
        if isinstance(texts, str):
            texts = [texts]
        self.tokenizer.src_lang = src_lang

```

```

        encoded = self.tokenizer(texts, return_tensors="pt"
                                   , padding=True, truncation=True).to(self.device)
    with torch.no_grad():
        generated_tokens = self.model.generate(
            **encoded,
            forced_bos_token_id=self.tokenizer.
                get_lang_id(tgt_lang),
            max_length=512,
            num_beams=5,
            early_stopping=True
        )
    return self.tokenizer.batch_decode(generated_tokens
                                       , skip_special_tokens=True)
def back_translate(self, texts):
    # Step 1: src      pivot (e.g., Sinhala      Bengali)
    pivot_texts = self.translate(texts, src_lang=self.
        src, tgt_lang=self.pivot)
    # Step 2: pivot      src (e.g., Bengali      Sinhala)
    return self.translate(pivot_texts, src_lang=self.
        pivot, tgt_lang=self.src)

```